NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

AGENT AND COMPONENT OBJECT FRAMEWORK FOR CONCEPT DESIGN MODELING OF MOBILE CYBER–PHYSICAL SYSTEMS

by

Curtis G. Adams

March 2018

Dissertation Supervisor Ronald E. Giachetti

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2018	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE AGENT AND COMPONENT OBJECT FRAMEWORK FOR CONCEPT DESIGN MODELING OF MOBILE CYBER–PHYSICAL SYSTEMS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Curtis G. Adams			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC) 6501 E. 11 Mile Rd. Warren, MI 48397-5000			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Military intelligent ground vehicle systems (MIGVS) have a wide variation in computationally controlled behavior logic that involves the interaction of both cyber and physical components as well as more typical systems engineering modeling needs and constraints. Current system concept design methods do not sufficiently address either the variation in cyber behavior linked to mission effectiveness or the integrated dependencies and interaction between the cyber and physical components. In this work, model-based concepts are developed to capture the required behavior logic as solution and assembly independent state-based agents and objects. These logical objects can be realized by alternative implementations and assembly aggregations, to include "human assemblies." The approach contributes a more thorough and robust model of the subject problem domain. These concepts include an agent and component object system data metamodel, supporting structural system classes, and state-based behavior concepts. The concepts are applied to a case study project to produce a solution-independent system concept design.

14. SUBJECT TERMS model-based system architecture, agent, component object, cyber-physical systems, concept design, autonomous convoy, intelligent behavior, architecture meta model, context model, logical model, trajectory, goals, agent logical object			**15. NUMBER OF PAGES** 303
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89) Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

AGENT AND COMPONENT OBJECT FRAMEWORK FOR CONCEPT DESIGN MODELING OF MOBILE CYBER–PHYSICAL SYSTEMS

Curtis G. Adams
B.S., Michigan State University, 1979
M.S., University of Michigan–Dearborn, 1999

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2018

Author: Curtis G. Adams

Approved by: Ronald E. Giachetti
Professor of Systems Engineering
Dissertation Supervisor

Raymond Madachy Associate Professor Systems Engineering	Bryan O'Halloran Associate Professor Systems Engineering
Marcus Stefanou Associate Professor Computer Science	John M. Reed Senior Operations Analyst U.S. Army TARDEC

Approved by: Ronald E. Giachetti, Chair, Department of Systems Engineering

Approved by: Orrin. D. Moses, Vice Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Military intelligent ground vehicle systems (MIGVS) have a wide variation in computationally controlled behavior logic that involves the interaction of both cyber and physical components as well as more typical systems engineering modeling needs and constraints. Current system concept design methods do not sufficiently address either the variation in cyber behavior linked to mission effectiveness or the integrated dependencies and interaction between the cyber and physical components. In this work, model-based concepts are developed to capture the required behavior logic as solution and assembly independent state-based agents and objects. These logical objects can be realized by alternative implementations and assembly aggregations, to include "human assemblies." The approach contributes a more thorough and robust model of the subject problem domain. These concepts include an agent and component object system data metamodel, supporting structural system classes, and state-based behavior concepts. The concepts are applied to a case study project to produce a solution-independent system concept design.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3D CAD	three-dimensional computer aided design
4D/RCS	four-dimensional real-time control system
ALO	agent logical object
ADL	architecture description language
BG	behavior generator
BDD	block definition diagram
C4ISR	command, control, communications, computers, intelligence, surveillance, and reconnaissance
CPS	cyber-physical system
DM2	data metamodel
DOD	Department of Defense
IB	interface block
IBD	internal block diagram
IoT	Internet of things
EBOM	engineering bill of materials
ICA	intelligent control agent
JCIDS	Joint Capabilities Integration Development System (JCIDS)
MA	mission agent
MBSA	model-based system architecture
MCPS	mobile cyber-physical system
METT TC	mission, enemy, terrain and weather, troops and support available, time available, and civil considerations
MIGVS	military intelligent ground vehicle system
MoC	model of computation
MOE	measure of effectiveness
MOP	measure of performance
OOAD	object-oriented analysis and design
PLS	Palletized Loading System
I/RSTA/EW	intelligence, reconnaissance, surveillance, target acquisition & electronic warfare

QA	quality attributes
SCRM	system context reference model
SE	systems engineering
SOA	service-oriented architecture
SP	sensor processor
SysML	System Modeling Language
WM	world model
WSM	world state model

EXECUTIVE SUMMARY

Concept design is a critical stage of systems engineering. A key objective of concept design is to model the system behavior independent of any particular physical solution, i.e., distinguish what a system needs to do from its method of doing. This allows alternative physical solutions to be assessed against how well each meets the intended behavior, usually within a trade space framework. This solution independent behavior modeling is referred to here as initial or logical concept design. The final concept design is the integration of the logical concept design with the selected physical concept and is used to support or govern detailed design and product development.

Each type of system, sub-type with specialized features, can have both common and unique behavior logic relative to each other. System types include weapon, command and control, business, enterprise, intelligent, information, autonomous, etc. Descriptive modeling techniques for logical concept design have emerged through practice and tend to support or favor certain system types over others. For example, functional-based design using Functional Flow Block Diagrams (FFBDs) emerged in support of mostly analog control of mechanical components in defense related applications, Business Process Modeling Notation (BPMN) in support of information management for business information systems, etc. FFBDs were augmented with data modeling techniques, such as data flow diagrams, as mechanical elements increasingly came under digital or computational control.

Cyber-physical systems (CPS) has recently emerged as a system type and can be defined as a system with computational control of physical processes (Lee and Seshia 2017), and are focused on the interactions or "the intersection of the cyber and the physical." Weapon systems have been evolving toward CPS for some time. As a mobile CPS (MCPS) they present an additional challenge or feature of computational control of physical processes that interact with a dynamic external environment or context. MCPS can be further typed as a military intelligent ground vehicle system (MIGVS) with specific additional features: a relatively complex context, synchronized execution within a command and control hierarchy, and goal directed context aware intelligence. These

additional features result in multiple forms of behavior that must be accounted for in the logical concept design.

The problem is current modeling practice for logical concept design lacks the necessary techniques to model all forms of MCPS and MIGVS behavior, such as cyber-physical, operational information management, and goal-directed intelligent. Without sufficient and relatively equal capture of all forms of behavior, the final concept design is likely to be sub-optimized along one or more of them. This research contributes an architecture modeling approach that integrates all forms of behavior into a single system descriptive model that is solution independent yet directly supports physical component selection and alternative system assemblies. The "component" selection and assembly alternatives enabled include the realization of intelligent behavior by human operator(s) in lieu of or in conjunction with machine computation, control and sensing.

The modeling approach is based on an overarching architecture concept data metamodel (DM2) as shown in Figure 1. This concept DM2 provides the overarching focus and framework for the supporting foundational terms and concepts. A MIGVS has a computational control logical hierarchy built on a base CPS layer, the latter is a required enabler for system intelligence. The logical hierarchy, like social hierarchies, are identified not by spatial proximity but by interaction (Simon 1962), in this case logical interaction. These interactions react and drive behavior in the form of world state change, both the system and the context. Key foundational concepts are:

(1) Performer and Context Objects—as indicated, the world from the system perspective is composed of these objects. They have attributes that determine their state and are abstractions of some underlying physical reality. The system performer objects react to events and attempt to create effects in the context.

(2) Goal-Based Trajectories are a desired or commanded path through a given world state space. Missions can be organized into a set of tasks which in turn can be organized into an ordered set of these desired trajectories. Mission orders or plans can be structured and stored as a set of successive world states with specific state and time goal measures.

(3) Agent Logical Object (ALO) is a system performer object based on a granular intelligent or human operator role. Each ALO stores an

assigned mission of goal-based trajectories and interacts with other ALOs via command and percepts. ALOs initiate behavior based on their world belief state relative to their goal state. The ALOs together comprise an intelligent logical control hierarchy above the base CPS layer.

(4) Horizontal Interactions—Interactions in a computational hierarchy can be distinguished between horizontal based on like data use at equivalent hierarchical levels, and vertical based on transforming the data (Shames and Sarrel. 2015) for use between hierarchical levels. The primary focus is horizontal interactions between hierarchically arranged ALOs. This hierarchy sub-divides the application level of computational hierarchy.

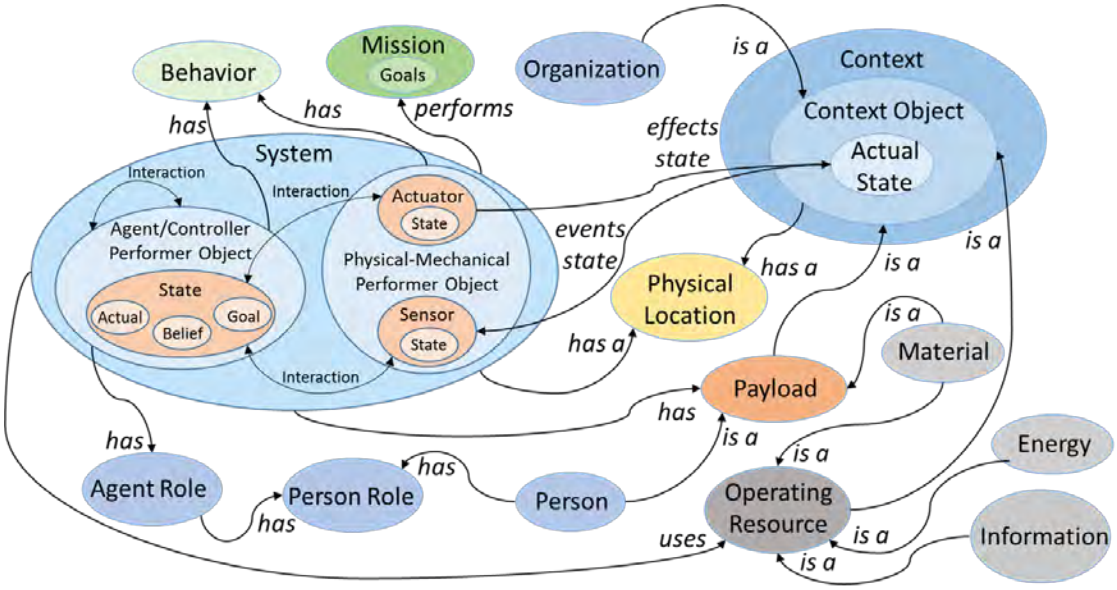

Figure 1. MCPS Concept Data Metamodel

The foundational concepts are modeled using the System Modeling Language (SysML). The system modeled concepts were used to support the generation of a case study system model also in SysML. The case study was based on a design reference mission of convoy following and design reference system of a Palletized Loading System (PLS). The system model consists of a set of performer objects based on an abstracted subset of the PLS physical hardware capabilities as well as a subset of two-person crew capabilities.

The system model was examined against the research questions. The system model generally is technologically independent, including a set of ALOs and sensors that could be realized by human or machine. All required forms of behavior are captured and sufficiently described. Operational and system behavior are integrated. Most of what could classically be described as operational behavior is incorporated into ALO behavior, mission assigned desired trajectories that they operate upon, and the interactions of ALOs. Most of what could classically be described as system behavior is within the CPS base layer. The system model is structured into a set of performer objects which enable direct link to component solutions, but have a level of granularity that would allow those component solutions to be aggregated into a variety of assembly approaches.

The MCPS concept DM2 and the supporting foundational concepts provide the necessary perspective and supporting framework to support generation of a system model that captures all the necessary forms of behavior. It also does so in a way that could facilitate physical component selection and trade space analysis. The ALO concept arranged in a hierarchy provides sufficient fidelity in terms of interactions and goal state yet retains it physical solution independence. A discovered benefit of this approach is the ability to conceptualize three types of knowledge or data (Evans et al. 2002) parametric, symbolic, and complex data structures such as arrays, lists, maps, images, etc. This data conceptualization would be an important part of the overall MIGVS conceptualization.

LIST OF REFERENCES

Evans, John M., Elena R. Messina, James S. Albus, and Craig I. Schlenoff. 2002. "Knowledge Engineering for Real Time Intelligent Control," *Proceedings of the International Workshop on Intelligent Knowledge Management Techniques (I—KOMAT 2002)*, Crema, Italy, Sept. 16–18

Lee, Edward A., and Sanjit A. Seshia. 2017. *Introduction to Embedded Systems, A Cyber-Physical System Approach*, Second Edition. Cambridge, MA: MIT Press.

Shames, Peter M., and Marc A. Sarrel. 2015. "A Modeling Pattern for Layered System Interfaces," *25th Annual INCOSE International Symposium (IS2015),* Seattle, WA. July 13–16.

Simon, Herbert A. 1962. "The Architecture of Complexity," *Proceedings of the American Philosophical Society*, 106: 467–82.

ACKNOWLEDGMENTS

I would like to acknowledge Dr. James P. Richardson, DCS Corporation, for his participation and the generation of the mobility portion of the case study model and diagrams. This was accompanied by discussions, in particular as regards various SysML modeling approaches, goal specification, state space representation, constraints, mathematical/difference functions, and complex data structures. These discussions provided valuable feedback, perspective, and insight that I used to refine the related foundational concepts.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

This research contributes to the logical concept design phase of systems engineering by integrating multiple forms of logical behavior into a single descriptive system model. The forms of logical behavior include operational behavior (a kind of business behavior), intelligent behavior that is goal-directed and context-aware, and the control part of cyber-physical behavior. The system model is solution independent, makes no assumptions about any human operators, and provides a holistic capture of the logical problem domain in a way that can directly support technology component allocation and system trade space analysis.

The system model is based on an underlying concept data metamodel (DM2) with foundational modeling concepts that support explicit modeling of the logic in terms of system objects, state behavior and interactions. Both the goals and the context associated with intelligent behavior can be captured in terms of objects and state. The approach of modeling system objects, their state, and their interactions, can be distinguished from a more typical systems engineering approach based on task or functional decomposition and interaction that is then mapped into some higher level system physical solution. The concepts are defined for the domain of mobile cyber-physical system (MCPS) in general and are applied to a more specific sub-domain, military intelligent ground vehicle system (MIGVS) in a case study.

A. BACKGROUND AND MOTIVATION

Cyber-physical systems (CPS) are defined by the National Science Foundation (NSF) as “smart networked systems with embedded sensors, processors, and actuators that are designed to sense and interact with the physical world” as well as human users (National Science Foundation [NSF] 2015). In CPS, the behavior emerging from the interaction between the “cyber” and “physical” elements of the system is critical. Physical elements are machines that interact directly with the physical world, and cyber elements are the computation, control and networks of the system. CPS can be further distinguished as having multiple types or models of computation. These in turn exhibit

what can be described as different types of associated behavior logic, to be defined here as cyber behavior or cyber controlled capability. The CPS behavior rests upon and/or is integrated with the physical behavior or physically driven capability. The cyber driven and physical driven capability are together realized by a set of technologies (i.e., computers, networks, software and machines). For clarity, the term component will be used when discussing the physical realization of technology for either cyber or physical capability.

Figure 1 shows the technology challenges that apply to defense systems (National Science Foundation [NSF] 2015). Weapon systems have a set of computational types, a set of machines that interact with the physical world, and an added set of physical constraints due to their mobility and other factors. This research focuses on MIGVS, a military ground system with autonomy levels, machine automation, and/or data automation significant enough to improve the mission effectiveness to crew size ratio relative to a comparable, non-intelligent system. This makes a reduced crew solution feasible or conversely can improve mission effectiveness of a vehicle with the same size crew.

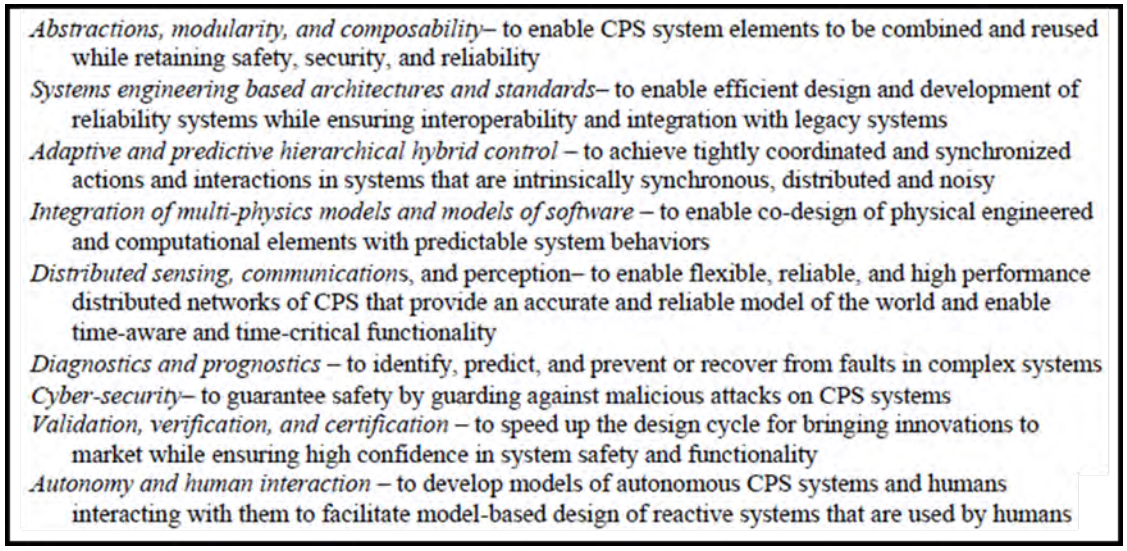

Figure 1. NSF Cyber-Physical System Technology Challenges. Source: NSF (2015).

A MCPS is likely to include four types of behavior logic (Douglass. 2004. 140–144, leveraged for first three definitions):

1. *Simple functional behavior* that transforms an input into an output without the use of memory or knowledge of previous input. Examples include direct flows of combinatory logic and mathematical functions, including control system transfer functions.

2. *State-based discrete-event behavior* that transforms discrete input in combination with previous input captured as state information to a discrete output. This is the behavior logic most closely associated with data or information processing that transforms information from one form to another, often in in response to a user input.

3. *Continuous behavior* that transforms continuous time input to a continuous time output where any previous input dependencies or state are also continuous. Examples include physical processes and "algorithmic objects" that transform a "stream of data" (Douglass 2004, 143–144).

4. *Intelligent behavior* exhibited by an agent or entity that is goal-directed (Russell and Norvig 2003. 44–50) and that can act on knowledge and perhaps understanding (Ackoff 1989). An intelligent *agent*, man or machine, is context-aware, meaning it can perceive and understand its environment and bring about an effect in that environment. For an MCPS, if one considers the crew or user as part of the system during its operation, then intelligent behavior can be realized as fully autonomous or as human-machine interaction. Intelligent behavior includes mission and task-oriented behavior driven by discrete events and that form the top level of an application control hierarchy. Intelligent behavior also includes "intelligent detection and control behavior" that sense or act in a dynamic environment and can be continuous time based. In a CPS, intelligent behavior represents a distinct type of computation.

CPS have also been described as "integrations of computation with physical processes" (Lee 2008). Figure 2 shows a MCPS viewed as the integration of "physical behavior" or processes and "cyber behavior" or computation. A set of physical mechanisms realize physical behavior and direct energy and/or transfer material to, from,

and within the system to achieve an effect in the external environment, or maintain some relative state in a dynamic environment. The mobility aspect of these systems increases the state properties of interest because even static properties in the external environment can now have a state relative to a moving system. A set of physical components realize cyber behavior and generate or transform signals into information to, from, and within the system and then perform computations on that information. The cyber components realize the four types of behavior logic as required to effectively control the physical mechanisms, manage its own information and data state variables, and make decisions about behavior. In the model, a human is a cyber component that acts through the physical mechanisms of the system and is considered part of the system or contained within the system boundary.

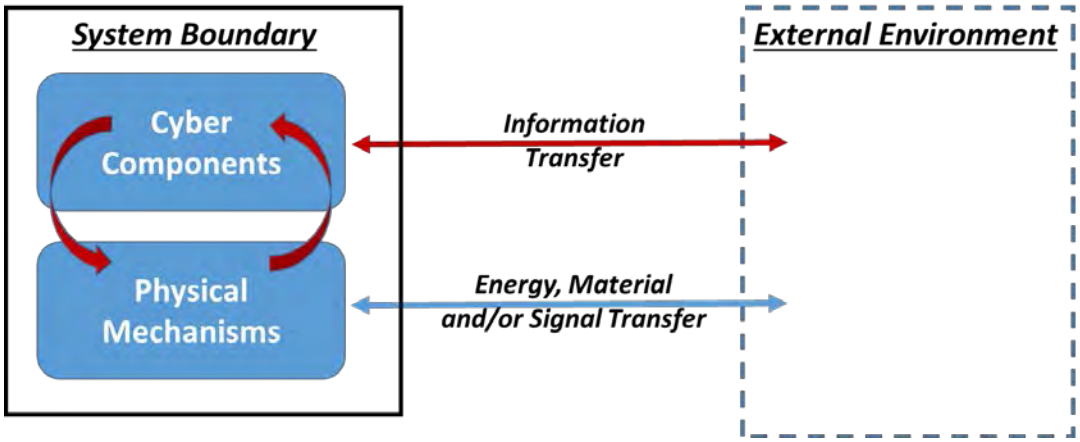

Figure 2. General MCPS Model

The challenge of realizing a MCPS with the full range of behavior logic described above is not only to sufficiently capture and integrate with the physical mechanisms, but do so within a larger systems engineering approach that can inform decisions about the entire systems architecture. The systems engineering includes selection of the components and mechanisms, determination of the overall internal/external structure and boundary, and achieving necessary performance within constraints. To accomplish this, the needed behavior logic must be captured solution independent and in a way that facilitates the assessment of potential solutions. What follows is a background as to how

these issues impact an MCPS architecture concept design. This will be viewed through the lens of the MIGVS domain.

1. An Historical Perspective on System Behavior Logic

Classical systems engineering methods for analyzing behavior logic, such as functional analysis and functional flow block diagrams (Blanchard and Fabrycky 2011, 86–93), emerged when systems were still largely mechanical and physical. The behavior logic is modeled as functions to transform inputs to outputs within a mission thread. Functions and functional flows are readily understood and can be reasoned about by multi-disciplinary teams, to include non-technical members. They are also inherently solution independent. The control by the functional flows represented could be realized by relatively simple devices such as servos and regulators directly linked to the physical mechanisms or machines.

The initial application of cyber technologies were first used to achieve advances in machine control, precision, and accuracy. This application resulted in improvements to thread performance of physical driven behaviors and fit within classical functional analysis, though functional flow analysis now had a greater emphasis on feedback control logic. However, as systems added more discrete software, automation, and/or autonomy to augment human operator capability, the resultant intelligent behavior has either an a priori allocation to human or machine execution, or presumes that this allocation can be addressed in software development after the hardware is rationalized and major trade space decisions accomplished. If the a priori allocation was not sufficiently analyzed or sufficiently captured, it cannot be adequately considered in the subsequent system engineering effort. In particular, the events and effects relative to any specified functional inputs and outputs will not be systematically and comprehensively linked.

The application of general purpose computing and associated software to weapons systems and other defense and non-defense systems, led to a common view of behavior logic between systems and software engineering under a formal method known as structured analysis and design technique (SADT) (Ross 1977). SADT was codified for the federal government under the Integrated (I) Computer Aided Manufacturing

Definition (DEF) modeling standards, particularly IDEF0 and IDEF1X. The latter being a data modeling standard considered necessary for software.

SADT or "algorithmic decomposition" (Booch et al. 2007) when applied to large scale discrete event software has been associated with issues of scalability, modularity, and a lack of intermediate forms (Booch et al. 2007). The behavior must be mapped to component implementations which often occur at higher level assemblies, and then further decomposed. A working model of behavior often does not occur until late in the development process and often subsequently to implementation decisions. If requirements change or the implementation is not decided correctly, it is difficult to isolate the change and consequently many different parts of the implementation can be impacted.

Object-oriented analysis and design (OOAD) methods emerged for large scale discrete software development efforts (Fichman and Kemerer 1992). The applications or domain logic is organized around classes/objects that encapsulate behavior and state and that interact to realize the system's behavior. These classes/objects help organize very complex logic in a systematic way, enable analysis of the system at a relatively high level of abstraction, enable modification and incremental increases in fidelity, and enable reuse of the objects to the extent that the domain logic is similar. It should be noted that SADT and OOAD are orthogonal methods to accomplish the same objective and cannot be used simultaneously to construct a system (Booch et al. 2007). An overarching OOAD "systems" methodology is exemplified by the practice surrounding the Unified Modeling Language (UML). UML evolved from OOAD via the Rational Unified Process (Rational 1998) combining concepts from James Rumbaugh, Grady Booch, and Ivar Jacobson. The architecture practice evolved from an original "4+1" view model concept (Krutchen 1995). As the practice has evolved, some of the names and meanings of these views has evolved as well. They are shown in Figure 3, and are defined in a way that better relates to an MIGVS context, but remain similar to their original form. The objects model "real world" entities in terms of classes that comprise the system. Secondly, the logic at the leaf level of classes and object can reflect relatively low levels of software code or an implementation and thus a direct instantiation of a software "physical" component

solution. Finally, software objects aggregate up in higher level software assemblies like programs, software configuration items, and executables. These elements constitute the "physical embodiment" of the software that are deployed to the hardware in the Deployment View. The one view in the "4+1" view model are use cases. Use cases can be described as a "goal-oriented set of interactions between external actors and the system under consideration" (Malan and Bredemeyer 2001). A use case provides a mechanism to determine critical functions as the system interacts with the external context, at least external "users." Use cases can be arranged in a hierarchy at various levels of detail or abstraction (Topper and Horner 2013).

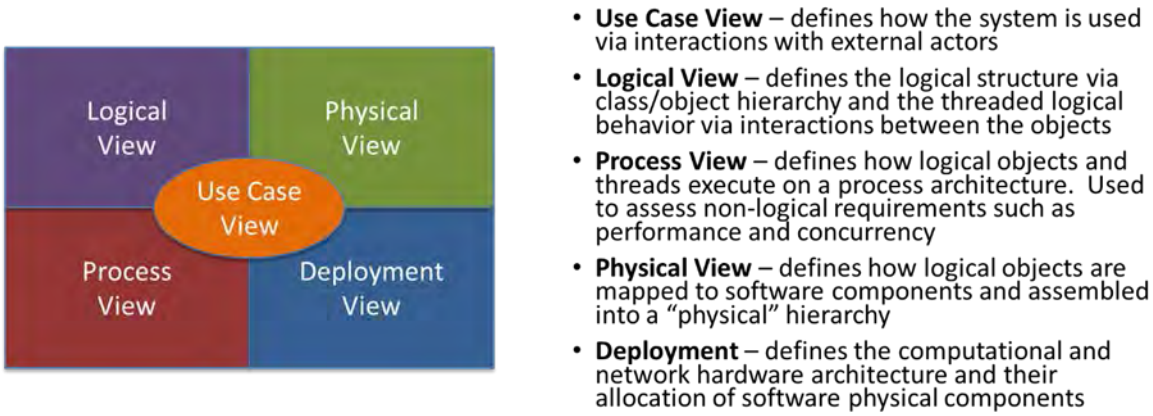

Figure 3. An OOAD 4+1 Architecture Model View.
Adapted from No Magic (2015).

As mentioned previously, OOAD has found significant application for large discrete-event software systems. From an MIGVS standpoint, consideration must be given to how the discrete event-based techniques of OOAD apply to its other forms of behavior logic and their subsequent capture in the system concept design. Adaptations can be addressed for simple functional and continuous real time behavior to support software design (Douglas 2004). However, within a systems engineering framework, the behavior and software are first rationalized as part of a larger system level abstraction with all forms of behavior logic as well as non-computational hardware and other system constraints to be considered. There is not equivalent consideration given to analyzing the

physical mechanical hardware nor its interdependency with the software as part of the larger system concept design. The basic premise of a use case reflects a bias toward transaction-based interfaces with users rather than effects-based external impacts of physical-mechanical hardware. OOAD techniques must rely on some other systems engineering activity to sort this out and decided that software of a certain capability is required as part of a larger system solution. MCPS need not only understand critical behavior in the context of goals, but must understand and reason about the goals themselves as well as any associated state space.

Information systems engineering and integration eventually emerged as a relatively separate discipline to design and build business applications. Areas such as information management, data base design, service-oriented architecture (SOA) and enterprise architecture, brought a new emphasis on business processes and business organization as a key consideration in the derivation of a system's behavior logic. Enterprise architecture in particular can be viewed as "a high level design of the entire business" (Giachetti 2010), to include the modeling of the business processes and the interaction with the external environment. In this sense, it represents an initial form of a kind of intelligent behavior modeling and another type of cyber driven behavior logic. The highest level of behavior is modeled to include its purpose or goal. The business processes have a hierarchy similar to functions that can go by different names: activity, tasks, steps, etc. When needed, the term task will be distinguished by the implied completion of work and can often be associated with a goal, whereas activity will refer to the more general abstraction of movement or energy.

The enterprise view of system architecture for military systems is exemplified by the Department of Defense Architecture Framework (DODAF) Version 2.0.2 (CIO DOD 2010) and is mandated by the Manual for the Operation of the Joint Capabilities Integration and Development System (JCIDS), 12 February 2015. It takes many concepts from enterprise architecture and SOA and applies them to the operational behavior of military systems. It provides unique emphasis on aspects important to CPS concept design: interoperability, operational behavior threads in response to events, and identification and integration of operational activities with system functions. As shown in

Figure 4, it also has a concept architecture data metamodel (DM2) that explicitly captures the architecture concepts so that these concepts can be better understood and reasoned about.

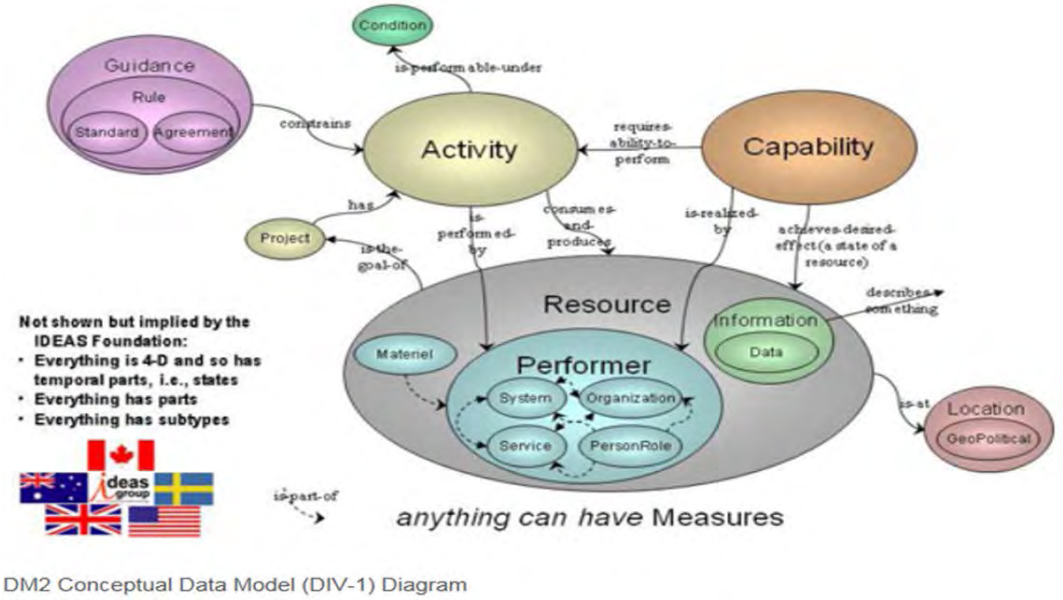

Figure 4. DODAF Concept Data Meta Model. Source: DOD CIO (n.d).

Though the DODAF is not a methodology, the viewpoints certainly have methodological implications as does its original establishment and evolution as an enterprise architecture. DODAF fundamentally supports a structured method, albeit a more complex one. Operational tasks activities are decomposed to system functions that can be further decomposed. The DM2 has supporting viewpoints that account for events and effects. However, although responding to events is accounted for in the DM2 in certain viewpoints, delivering effects the context is not formally modeled beyond "friendly" systems or organizations and interoperability concerns. Though there is a Services Viewpoint that includes measures of effectiveness (MOEs), there is no other viewpoint that would include depicting any context MOE that is not a resource. A goal is viewed as a capability which in turn is the desired state of a resource. In that sense, the context is more limiting than that of use cases. The interoperability and service focus and enterprise background also suggest a human transactional bias and world effects are not addressed. Finally, though it is stated that everything has states, it would seem that the

state of an activity means something different than the state of physical thing or software object.

Enterprise systems have an underlying information-based system model (i.e., users in a business office environment interacting via work stations connected by a network and supported by servers). Military command and control looks less like a standalone information system as the unit under consideration gets smaller and smaller and more directly interacts with a physical dynamic environment. Though not real time, timing is a factor in the behavior of these small unit C4ISR systems which makes large scale data base approaches less feasible. Specialized performance and constraints to synchronize and coordinate effects must be considered. However, there are some useful concepts that can be leveraged into an MIGVS architecture. The concept of modeling operational or business behavior logic as part of the overall architecture is particularly critical, especially if that behavior is to be realized whole or in part by the system. This is the case for the task-oriented part of intelligent behavior, even if it is only supporting information processing within that behavior. The concept of a node as an abstraction of a computational element within the architecture could be well utilized within a systems engineering framework, though its definition of "where information is processed" would have to be expanded to include managing work. The concept of mapping operational activities to system functions is a realization that there is a complex logical hierarchy that executes within a MIGVS.

Military systems perform or support tasks with a goal or end state to be achieve by work. This is true even for a C4ISR, particularly for a forward unit that is using interoperability and synchronization to achieve a specific mission objective or purpose. Task and mission behavior logic has always been critically important to a weapon system, particularly when we consider the human operator as a cyber component in the execution of that logic. In the sense of making the human an element of the system, a ground vehicle weapon system has always been a MIGVS. Until the introduction of on-board digital and software components to process information, the human interaction with the system was typically treated as a separate concern and mostly a design activity distinct from the system architecture definition. This information processing can be

utilized not only to aid the execution of unit level command and control, but can also be utilized to aid realization of system mission and task execution. The mission and task logic form a hierarchy of logic when integrated with machine behavior can accomplish work or goals beyond information processing. It is the realization of cyber-physical behavior.

A good example of realizing a system cyber-physical behavior is the hunter-killer capability (NGAUS 2014) within the Abrams Main Battle Tank. The hunter-killer capability was realized with the addition of the following information acquisition and processing components: an independent target acquisition suite for the commander, a commander's display and supporting processor, a gunner's display and supporting processor, a network to exchange information between commander and gunner, and software. These components in concert with the commander and gunner achieve a higher rate of target acquisition and kill under certain offensive and defensive operational scenarios. The increase in effectiveness relies on integration of the weapon and mobility physical-mechanical components as well as low-level physical behaviors such as move and shoot, it is achieved without any improvements to those capabilities. Furthermore, at a high level of abstraction, the commander and gunner do what they have always done in terms of coordinating target acquisition and firing priorities. The addition of information and processing components improved the coordination or control between the commander and gunner beyond direct human interaction. An increase to a measure of effectiveness (MOE) was achieved with the additional cyber capability as an intermediate level of control between the existing higher level cyber-capability (i.e., commander and gunner), and the lower level cyber-capability (i.e., direct control of the physical-mechanical components).

Any MIGVS has at least the same hierarchy of cyber capability just described, more if a given human operator is abstracted into a set of objects based on roles. Roles include managing the mission overall, performing primary tasks such as driving, and performing support tasks such as obstacle or threat detection. Each role-based object has associated MOEs and are supported by system components with measures of performance (MOPs). Mission effectiveness is supported by task effectiveness, which in

turn is supported by system performance and then technological performance (Spero et al. 2014). There is a hierarchy of cyber control with a hierarchy of MOEs/MOPs. Measures of mission effectiveness can be linked through the hierarchy to cyber components, such as the aforementioned information and processing components. (Badger et al. 2013).

A given MIGVS needs to achieve a certain operational effectiveness whether it has a full crew complement, full autonomy or something in between. The crew can be considered part of the "product set," and crew size, including zero, as part of the trade space in system concept design. Operational tasks need to be identified and allocated to the crew and/or an autonomous technology solution. Various combinations of these "products" need to be assessed for their impact on cost and operational effectiveness. These products must also be combined with more "standard" software products utilized for information management and other system logic and also assessed as part of the trade space. This requires a view into the product set that can be directly linked to the behavior and attributes and not biased toward a physical assembly and containment solution, to include a human operator as a particular type of physical assembly

It is not straightforward to assess the operational effectiveness with MOEs indistinguishable from crew versus technology solutions. The U.S. Army has much doctrine, task lists, field manuals, and unit training guides that describe what tasks the crew are to accomplish. Quantitative measures relative to the task execution along with their quantitative contribution to overall mission effectiveness are not as well specified. Mission and task MOEs are scenario dependent with variables of mission, enemy, terrain and weather, troops and support available, time available and civil considerations (METT TC) (ADRP 3–0, 2012). METT TC variables have a probability of occurrence with a certain degree of concurrency relative to each other for any given scenario. Event profiles (i.e., the probability and frequency of event occurrence), are not typically available and must be estimated to determine the system response, to include the crew, as well as measure its effectiveness. The system response in turn has an internal hierarchy of response dependencies through the product set that contributes to task execution.

Agent-related methods is a way to model intelligent behavior. The increasing emphasis on autonomous systems are likely to lead to widespread embedded agent application in the near future. Agent approaches are used currently in various simulation tools to support MIGVS analysis, but most simulation-based approaches are focused on the interaction of a large number of simple agents for domains that fit a more theoretical definition of a complex system. If one considers the "as is" physical embodiment of the agents as crew members that interact as part of the system, a MIGVS has fewer but higher level fidelity agents. The focus here will be on agent methods for "systems" that more closely resemble these MIGVS agents. Like most of the previous behavior logic review, these agent methods find their lineage in software. It is worth noting here that a simulation of a system is essentially a simulation of the behavior of an autonomous vehicle system, except for perhaps scope and fidelity. If the behavior logic was "perfectly" or "fully" simulated, then all the necessary behavior logic to realize an autonomous system would be defined.

Agent-oriented software engineering (AoSE) has only recently emerged from a "nascent field of research" to some areas of application, such as industrial agents and other multi-agent systems (Tveit 2001). Most AoSE techniques recognize similarity and distinction between objects in OOAD and agents in agent-oriented analysis and design (AOAD). Some methodologies like Gaia (Woodbridge et al. 2000) require a more significant departure between how agents are defined and modeled versus objects. Other methodologies, such as Multiagent Systems Engineering (MaSE) (Deloach et al. 2001), view agents in terms of extensions or specializations of objects. Supporting techniques like agent UML (AUML) (Odell et al. 2000) see leveraging object techniques as "risk reduction" for the relatively new agent technology.

The observable behavior of an MIGVS as a black box could be considered to be the same whether it is fully crewed or fully autonomous. As discussed previously, the crew itself can be modeled as agents where each crew member is decomposed into multiple agents based on role decomposition. Multi-agents combine within and between systems to form larger hierarchical unit that performs task-oriented operations as considered in the link of agent development to organization theory (Argente et al. 2006).

As such, logically linking agents and objects holds the promise of linking military units through the MIGVS to relatively granular role-based agents that interact with non-agents objects and entities. This should enable the integration of all forms of behavior logic from high level tasks to physical interactions with the environment, at least at some level of abstraction.

2. Model-Based System Concept Design

System concept design has been described as the "most important phase" in system development and one that translates a problem into a need and then a preferred solution (Blanchard and Fabrycky 2011, 56). Concept design includes a solution independent behavior design that meet operational needs as well as the physical concept of the preferred solution. Solution independent part of concept design must be done in a way that can be translated into more detailed behavior and physical design. Classically, concept design is done by analyzing the system behavior using FFBDs and then mapping or allocating portions of the behavior to some high level representation of the system hardware. Increasingly, the concept design phase is realized within a model based systems engineering (MBSE) is increasingly used to define a descriptive model (NDIA 2011) to support concept development (INCOSE 2007). Concept design supports embodiment design in a product design and development process and includes 3D CAD modeling (Arunachalam, Prakash, and Rajesh 2014). Mass properties and physical mechanical behavior can be conceptualized in a 3D CAD language, but not higher levels of behavior. MIGVS concept design requires a 3D CAD concept model integrated with ability to model higher levels of behavior.

SysML, in particular when coupled with the Object Oriented Systems Engineering Methodology (OOSEM) (OMG MBSE Wiki 2011), can be thought to provide some support to SADT and OOAD methods. Any structural abstraction can be modeled as a block extending the notion of a class beyond software. SysML also supports use case modeling to derive critical functions in a goal-oriented context, direct linkage and traceability to requirements, and behavior modeling techniques found in UML (i.e., activity, sequence, and state machines). SysML also has concepts for both

functional and logical architecture modeling distinct from a physical implementation. However, OOSEM utilizing SysML in practice generally follows a SADT approach for logical analysis with the addition of use cases and a facility for mapping to logical objects. Functional flows are modeled in activity diagram and data flows in sequence diagrams. It is seeks to combine structured and object-oriented analysis (i.e., combine two orthogonal techniques to simultaneously construct the same system) (Booch et al. 2007, 22). OOSEM seems to be closer to an object oriented technique if the logic is relatively simple (e.g., a logical hierarchy of one or two levels). When the logic is more complex, the bias is toward a complex functional architecture with one level of abstraction object logic above implementation as a facility for mapping (Hart 2015). Like the use of FFBDs, the tendency also is to focus on the control of physical mechanical behavior and assume higher level logic, if needed, can be deferred to a later stage software design.

Concept design of a ground vehicle system, in general, proceeds much like any weapon system concept design within a Department of Defense (DOD) acquisition and JCIDS based development. An initial set of desired operational capabilities is supported by trade space exploration, technology development, continued operational analysis, system requirements development, system analysis, and perhaps some system demonstrations. Typically, the ground vehicle community generates one or more physical concept designs in CAD models as an aid to these activities. The output is a selected feasible system concept, a finalized operational capability document, a system performance specification, a system cost estimate, and an acquisition approach. The concept design then matures toward a preliminary design or the full embodiment of the system architecture and details the interfaces between products. However, almost by definition, the major trades that determine the system architecture and identify the major products and technologies are usually completed with the concept design.

The cost estimate usually conforms to a standard surface vehicle work breakdown structure (WBS) (MIL-STD-881C Appendix G 2011). A portion of the WBS, shown in Figure 5, represents a product view of the system. It is meant to form a common, repeatable, and relatively granular domain structural logic of the systems products or

technology. The product structure is instantiate with specific technology solutions. For instance, Suspension/Steering would include considerations of track versus wheels; active, semi-active or standard suspension; braking systems, etc. Similar considerations can be done for other product groupings such as Survivability, Communications, Fire Control, etc. These technologies are then "integrated" into subsystems, sub-assemblies, major assemblies, etc., to form a system concept to meets its performance requirements and constraints. The system concept evolves based on performance and constraint analysis until the most feasible concept is determined. The CAD model is used to generate a high-level engineering bill of materials (EBOM) used for cost estimating. The most feasible concept is used to build an acquisition program budget.

G.3 WORK BREAKDOWN STRUCTURE LEVELS

WBS #	Level 1	Level 2	Level 3
1.0	Surface Vehicle System		
1.1		Primary Vehicle	
1.1.1			Primary Vehicle Integration, Assembly, Test and Checkout
1.1.2			Hull/Frame/Body/Cab
1.1.3			System Survivability
1.1.4			Turret Assembly
1.1.5			Suspension/Steering
1.1.6			Vehicle Electronics
1.1.7			Power Package/Drive Train
1.1.8			Auxiliary Automotive
1.1.9			Fire Control
1.1.10			Armament
1.1.11			Automatic Ammunition Handling
1.1.12			Navigation and Remote Piloting
1.1.13			Special Equipment
1.1.14			Communications
1.1.15			Primary Vehicle Software Release 1…n
1.1.16			Other Vehicle Subsystems 1…n (Specify)
1.2		Remote Control System (UGV specific)	
1.2.1			Remote Control System Integration, Assembly, Test and Checkout
1.2.2			Ground Control Systems
1.2.3			Command and Control Subsystem
1.2.4			Remote Control System Software Release 1…n
1.2.5			Other Remote Control System 1…n (Specify)
1.3		Secondary Vehicle	

Figure 5. Ground Vehicle System "Product" WBS.
Source: MIL-STD-881C Appendix G (2010).

Like many other systems engineering techniques, the product view of a WBS first emerged when systems were predominantly composed of physical mechanisms and some associated control system technology. With the increasing embedded application of digital/information technology or higher level cyber components, the following problems begin to arise:

a. The granularity of the physical mechanisms as compared to the granularity of cyber components are not equivalent, except for direct control that can be associated with the physical mechanism (e.g., fire control). The higher level cyber products are allocated to relatively large "buckets," such as vehicle electronics and Primary Vehicle Software #1. Software #1 provides little insight into what that software does, how it would impact a trade space, and is often left to large design efforts. It is roughly equivalent to describing the physical mechanisms as Physical Assembly #1, Physical Assembly #2, etc.

b. Products previously considered outside the system boundary, such as certain training, maintenance, and test components, may now be included wholly within the system boundary or split between on-board and off-board capability. If included in the system, the connection to any related off-board cost is lost. It also tends to skew cost comparisons between primary vehicle systems that have this capability embedded versus those that do not. For example, a system may have embedded training and/or embedded diagnostics/prognostics that wholly or at least in part replace off-board training and maintenance equipment. Presumably off-board costs would go down and primary vehicle costs would go up. An effective trade space analysis would have to account for the capability and cost of the embedded components together with the off-board "components" and determine the best combination. When compared to other primary vehicles costs and capabilities, it would not appear to be more expensive with little insight into the higher cost, yet it might in fact improve training and maintenance and therefore the readiness of the system.

c. Much of the higher level cyber products are developed separately from the system, provided as government furnished equipment (GFE), and not included in the WBS. The GFE often overlaps with WBS elements only generally described as

electronics and software. The overlap results in a more costly and sub-optimized concept design in terms of integration constraints which are difficult to identify let alone rectify. Attempts by the system developer to eliminate duplication and improve GFE integration usually get reflected as a cost to the system. The developer's system appears more costly when compared to a similar system with sub-optimized GFE integration.

d. Task-oriented behavior allocated to cyber components are identified as system costs. Task-oriented behavior allocated to the soldier are not identified as system cost. A system with cyber components with equivalent or better operational effectiveness is at a cost disadvantage as long as the cost of the soldier is zero. However, the cost of the solider in terms of housing, training, etc., is not zero.

All of the issues above can be addressed or at least mitigated, if the cyber components or objects can be structured with similar granularity as the physical-mechanical components. Role-based agent objects like mission agent, driving agent and obstacle detection agent provide object granularity into the system's logic. These objects can then be combined with a compatible set of physical objects similar to the existing set in 881C. The crew can then be considered part of the physical architecture solution and used for certain cost and performance comparisons.

Programs need the ability to consider the crew relative to the system product set or to abstract the crew into a "technology neutral" product set. Agents can be utilized to capture the crew behavior to a fairly granular degree appropriate for concept design while remaining technology neutral; that is, the agents can be allocated back to the crew with associate training standards, realized by cyber technology, or both. Agent objects integrated with non-agent system objects determine the full system response. Soldier workload and computational resource loading can be assessed with the allocation of agents and objects to technology. The right side of Figure 6 shows an integrated operational and system model for behavior formed when the technology neutral object model is extended with technology selection. The left side of Figure 6 shows an information translation loss from operational model to operational requirements narrative to system requirements narrative to system model. Information translation loss can be

avoided if the requirements are integrated with and extractable from an integrated operational and system behavior model.

The integrated operational and system behavior model relates primarily to task and functional requirements. Task requirements are often unspecified because they are presumed to be executed by an operator. Other requirements may be better expressed in other types of models or as narrative. System mass property requirements are best captured in a CAD model. There may be functional or constraint requirements that reflect standard practice and/or lessons learned that would not be revealed as a decomposition of an operational behavior. Also, narrative requirements will likely always be needed for non-technical stakeholders, such as program managers and contract officials. Therefore, an integrated and operational and system behavior model, like other operational and system models, are best viewed as a companion to a narrative set of requirements.

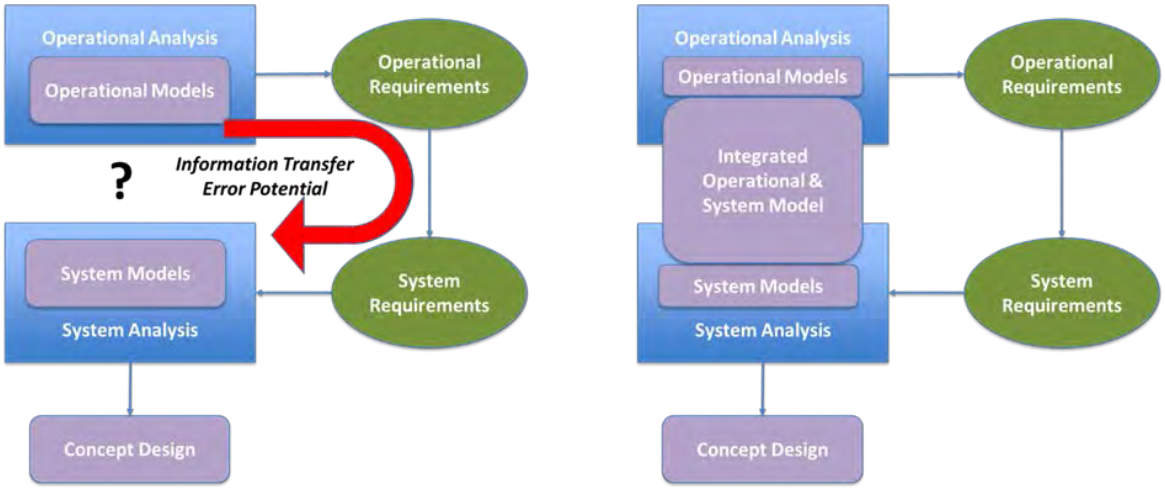

Figure 6. Integration of Operational and System Models

Logical concept design is a solution-independent model of the problem domain that describes what the system has to do. Logical concept design is typically associated with defining a functional architecture (Blanchard and Fabrycky 2011, 93). As Blanchard and Fabrycky explain, concept design is complete when the system architecture is defined (i.e., the functional architecture is mapped into a physical architecture supported

by trade-off analysis). Completion of concept design leads to the preliminary design in a systems engineering development process. Blanchard and Fabrycky acknowledge the completion of concept design with a concept design review. Concept design completion would have to occur as part of the system functional review (SFR) in current systems engineering practice for the DOD (Department of the Navy [DN] 2015).

A logical concept design that integrates operational and system behavior ideally initiates as part of the earliest formulation of a program (i.e., coincident with or shortly after operational need definition and early operational analysis). The logical concept design would mature as needed through early program phases and formal reviews (e.g., system requirements review [SRR]). The use of the logical concept design in the development of the system architecture enables trade-off analysis to address physical architecture impacts to both the system and operational behavior. The resultant physical architecture would also include both hardware and software architectures. Greater agility is therefore enabled from operational need definition through final concept design, Standard reviews such as SRR and software specification review are still held as needed to mark firm decision points or baselines and would include a concept design maturity assessment. Ideally, a more focused concept design review would be held to finalize concept design as opposed to being a part of SFR. In either case, the preliminary design phase can proceed as before.

The impact of cyber components and cyber capability can also be revealed anecdotally. An F-35 pilot, Lieutenant Colonel David Berke (Martin 2014), after agreeing that the F-35 does not fly any faster, or maneuver more sharply, than other planes, stated that: "Those are metrics of a bygone era. Those are ways to validate or value an airplane that just don't apply anymore…The biggest big deal is the information this airplane gathers and processes and gives to me as the pilot." Though there is perhaps an equal risk in underestimating the value of the system's mechanical performance to the overall system behavior, it is certain that a system with several million software lines of code has a cyber-intensive behavior component that is now interdependent with a tradition weapon system's physical-mechanical intensive behavior component. A system concept design

for this type of CPS must be capable of simultaneously modeling both types of components, their interdependence, and their respective impact on mission effectiveness.

3. Summary

MCPS, particularly when we include the operator, reflect key features of CPS: cyber and physical intensive, cyber and physical interdependence of behavior and attributes, and multiple classes of computation. The behavior or the anticipated behavior of the embedded operator(s) in a MIGVS can be extracted out into solution independent logic objects based on roles. The role-based objects are mission and goal oriented and arranged in an interdependent hierarchy of logic or cyber behavior.

SE methodologies or implicit methodologies in architecture frameworks or a system language like SysML, have brought certain advantages and disadvantages to system concept design, as summarized in Table 1. MIGVS behavior concept design requires a superset of these features:

(1) Solution Independence

(2) Integrate operational and system behavior

(3) Model intelligent behavior

(4) The ability to directly reason about state

(5) Component objects that can directly translate to component implementation

(6) Equivalent consideration to human, software, hardware and physical-mechanical behavior and their component abstractions

(7) Enable component assembly alternatives

Table 1. Concept Design Methods and Considerations

Method	Advantages	Disadvantages
Structured System Analysis & Design (e.g., FFBD)	1. Solution Independent 2. Straightforward reasoning of functional behavior relative to inputs and outputs.	1. Not directly translated to implementation leading to concerns about modularity and scalability 2. Imprecise level of decomposition relative to component allocation 3. Behavior often not fully understood until after an allocation to a set of high level assemblies 4. No direct reasoning about state, operational behavior, event/effects, or intelligence
Software Object-Oriented Analysis and Design	1. Can directly reason about state 2. Classes/objects lead to direct component implementation facilitating modularity and scalability 3. Assembly independent 4. Use cases generate critical functionality relative to the system context and can be goal oriented.	1. Not a whole system approach– just software 2. Limitations of use cases: simple context and no explicit support for modeling effects, intelligence, and concurrency/interdependency
Enterprise Analysis & Design	1. Explicit meta model for architecture concepts and relationships 2. Support business or operational modeling and associated events 3. Supports direct linkage of operational activities and system functions.	1. SSAD approach only more complex with many of its disadvantages 2. Human transactional and interoperability centric versus goal and effects based 3. No direct support for modeling context, effects and intelligence
SysML Based MBSE Considerations (e.g., OOSEM)	1. Concept of a block supports flexibility of class definitions beyond what is implemented in software 2. Provides some facility beyond SSAD for mapping behavior to implementation 3. Many SSAD advantages 4. OOAD use case advantages	1. Mix of orthogonal methods: SSAD and OOAD and has SSAD disadvantages for complex logic 2. Bias towards hardware and low level physical mechanical behavior with cyber behavior deferred to later development 3. OOAD use case disadvantages.

The systems engineer of CPS now requires methods to capture the full range or heterogeneity of behavior discussed here and its relation to physical aspects of the system. These methods must address:

(1) Capture of intelligent behavior

(2) Specification of operational tasks not associated with unit command

(3) Task allocation to operator versus machine

(4) Trade off analysis of cyber and physical capability and components

(5) Sufficient logical behavior analysis prior physical architecture decisions

(6) Decomposition and allocation of MOEs/MOPs for behavior

(7) Descriptive methods that inform product development with sufficient behavior comprehensiveness.

These methods have to be model based given the complexity and multi-disciplinary nature of CPS and the emergence of model based methods in both systems engineering, software engineering and product development. Additionally, the behavior logic is better addressed in models rather than with requirements narrative. These methods will also need to enable the allocation of cyber behavior models to support design as well as form a system architecture and decision baseline at a system concept level of abstraction.

B. PROBLEM STATEMENT

MIGVS have levels of CPS behavior logic with features that are not addressed by current behavior modeling methodologies and approaches. As a result, the system's higher level behavior, including intelligent behavior, is not effectively captured and therefore sub-optimally considered in concept design.

C. RESEARCH SCOPE AND OBJECTIVES

The purpose of this research is to define and demonstrate approaches and concepts to effectively capture MIGVS behavior logic for concept design. The research proposed an overarching architecture concept data metamodel (DM2), shown in Figure 7, that identifies key architecture structural and behavior concepts.

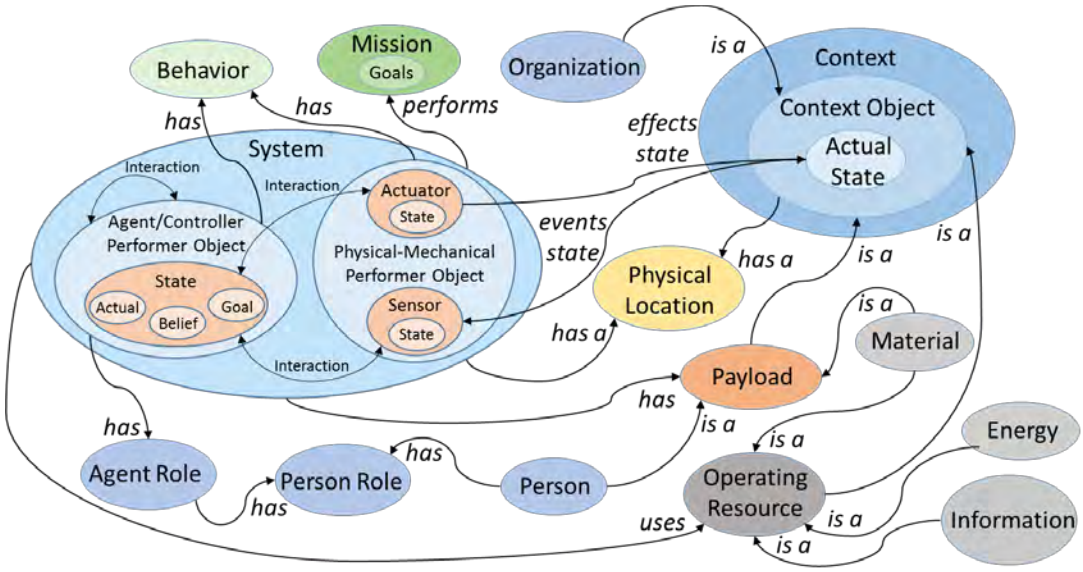

Figure 7. MIGVS Architecture Concept DM2

The key structural and behavioral concepts are as follows:

(1) System Performer Objects

(2) Context Objects

(3) System Connected context objects like payload and operating resources

(4) Missions defined by Goals that reflect a desired trajectory through the state space

(5) Agent Logical Objects that are a specialized type of performer objects

(6) Behavior as state change driven by interactions between objects

As in software OOAD, objects can be organized and analyzed as class abstractions. Unlike OOAD, these classes are not abstractions of software code, but are abstractions of any physical object, that may include software code. The classes are logical concepts that enable system concept analysis and design. The MIGVS DM2 is supported by the following archetypical classes: Context, Performer, Agent, Goal, Payload, and Operating Resource.

These classes can be detailed and decomposed for a given application or domain, and used to construct a system and context model. Decomposition can be done to the

point of component types, but not dictate a specific solution or technology. Specific missions and goals can also be decomposed to reflect desired behaviors expressed in terms of context and/or performer object states. Performer and context objects change state through object to object interactions, including internal system object interactions and system to context interactions that represent events and effects. The model once fully developed would represent the initial concept design. Specific solutions can then be directly instantiated from the leaf level objects of the initial concept design. Different alternatives can be considered subject to different constraints and trades conducted and aggregated into physical assemblies, including "human assemblies," much like transitioning from the logical view to the physical view in software OOAD. In this case, the physical view would result in the final concept design.

The structural and behavior concepts just described are detailed and applied to a case study and an initial concept design of a PLS. The behavior of a manned PLS is abstracted into a set of implementation independent interacting objects. The mission will be a convoy and the system a follower system of that convoy. The case study artifacts will then be analyzed against the original concepts and research questions and conclusions and recommendations drawn.

D. RESEARCH QUESTIONS

Can the intelligent MCPS concept DM2 and its supporting foundational concepts be used to generate a system logical concept design behavior model that:

(1) Is independent of specific technological solutions, to include any human operator(s)?

(2) Gives equivalent consideration of functional, state based discrete, state based continuous, and intelligent behavior?

(3) Integrates operational and system behavior?

(4) Enables direct translation to component solutions with scalability and modularity?

(5) Is independent of a particular component assembly approach?

E. RESEARCH VALUE AND METHOD

This research adapts and advances systems engineering and MBSE methods to address cyber concerns. The research develops a framework that enables cyber concerns to be addressed equivalent to physical-mechanical and overall system concerns as part of system concept design. Use of the framework results an early component-based model of behavior that can integrate with other components and capability into a trade space. This model of behavior can also inform or provide a specification baseline for embodiment and/or configuration item product design beyond an allocated set of requirements expressed as narrative. Many emergent systems can be classified as MCPS, including most DOD weapon systems, and could leverage these concepts.

Much CPS development and research starts with the assumption that a CPS system is needed and there is sufficient knowledge of the requirements or desired behavior. This research will enable a more definitive determination of the value of a CPS system, its scope, and its needed behavior and capability. More alternative approaches to CPS system composition can be assessed, including that a CPS approach may not be worth pursuing. It also incorporates consideration of the human role and human intelligence into the CPS design that includes an agent as a potential model of computation which may warrant unique considerations.

The research method is design science and is patterned after design science of information systems. Information systems design science research can be distinguished from empirical and experimental research as seeking to shape the existing world versus explaining or describing the world (Iivari, 2007). More specifically design science can be distinguished between a paradigm of behavioral science that "seeks to develop and verify theories that explain or predict human or organizational behavior," and a paradigm of design science that "seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts" (Hevner, et al. 2004). Several design science research methodologies have been postulated to provide a framework and rigor for design science. The similarity of an information system design and an information-intensive MCPS concept design, warrants a similar design science method. A

design science research method based on an information system (Peffers, et al. 2008) was selected and the "activities" modified for NPS SE dissertation requirements as follows:

a. Problem Identification and Motivation

Conduct the background investigation and literature search necessary to specifically state the problems and identify the value if the problem is solved. This is captured in Dissertation Sections I.A and II. The value of the research in the context of the literature search is defined in Section I.E.1.

b. Define Solution Objectives

The solution objectives and the expected value should be "inferred rationally" from the problem statement, background and literature search. The objectives have been defined by the research objectives or hypotheses and the research questions as captured in Section I.C and I.D.

c. Create the Meta-artifacts

Information systems design research strategy can be described as building a meta-artifact (Iivari 2015), where the meta-artifact is "a general solution concept to a class of problems." Iivari further points out that an innovative meta-artifact is the outcome of the research and must be built as opposed to simply evaluated. This approach to research will be used, except that the research applies generally to systems or systems engineering versus information systems, with the particular class of problems being MIGVS with some extension to MCPS. The meta-artifacts are the Concept DM2, the "Agent and Object Oriented Behavior Model," and the underlying concepts and class archetypes. These concepts will be fully developed and described in Section III.

d. Generate the artifact from the meta-artifacts

A particular instance of an initial system concept design will be generated from the meta-artifacts as indicated in the case study description. An agent and object oriented behavior model of the "as is" behavior of the current two-persons crew operated PLS system will be developed. This will represent a technologically neutral view of the "as is"

logical behavior. This “as is” behavior can be mapped to either human of machine solution for all forms of behavior.

e. Demonstrate and verify with a design artifact that the meta-design addresses the problem and meets the solution objectives

The preferred PLS concept is the demonstration design artifact. The contribution of the meta-design toward defining the PLS logical concept design and addressing the MCPS defined problems, research objectives, and research questions will be analyzed and captured in Section IV. The analysis will seek to determine:

(1) Whether the concept design initial agent and object oriented model was both technologically neutral and a model of the understood operational behavior.

(2) Whether the all forms of behavior are represented in the model.

(3) The overall usefulness of the meta-design in generating the system concept design

f. Interpret and Communicate the Results

The analysis will be further interpreted and conclusions and recommendation drawn. The final dissertation will reflect the communication of the results as well the overall approach. The researcher will create and refine the meta-design concepts. The researcher will generate the case study concept design model alternatives, except for the mobility model which will be generated by the project team. The case study models and some of the meta-design will be generated in SysML. The project team will also collect and/or generate:

(1) Operational and requirements source data

(2) Convoy mission thread analysis

(3) General SysML modeling techniques and procedures

(4) Comment and feedback on the researcher’s meta-design and case study SysML model

Original source data will be current convoy doctrine, draft CDDs, standards utilized in requirements, a MOE/MOP framework (Badger, et al. 2013) an external context model (JC3IEDM 2007), and SysML modeling guidelines. Some of the source data is Distribution D or FOUO. The resultant project concept model is also likely to be FOUO and perhaps some of case study analysis. However, the research concepts and most aspects of the analysis are not expected to be restricted.

THIS PAGE INTENTIONALLY LEFT BLANK

II. PRIOR WORK

Relevant prior work can be divided into two parts: 1) general and related domain system architecture and system modeling approaches and methods that look to integrate at least some aspect of behavior logic with physical mechanisms and the physical embodiment of the system, and 2) specific work related to CPS or MCPS system architecture. These approaches will be assessed against the three research questions identified previously.

A. GENERAL SYSTEM ARCHITECTURE APPROACHES AND METHODS

Since defining the behavior logic and its subsequent integration into the physical embodiment of the system has been concept design goal of most any view of systems engineering, it is important to understand how this is address or not address in general system architecture approaches and methods. These are reviewed in three types: domain models, model based system architecture, and architecture and modeling design languages.

1. Domain Models

As mentioned previously, concept design now requires views into products beyond a product hierarchy based on physical assembly and containment relationships. Views into the product set in terms of their logical and executable relationships are now required. Domain modeling can be described as a way to capture concepts from the problem domain and build a common language for communication across a project or enterprise relative to those concepts. As such, it not only has embedded architecture concepts often in terms of logical layers, entities, and their relationships, but can guide the specific formulation of a given project's architecture. A domain model can be a relatively general reference model for a specific focus, such as the open system interconnect (OSI) model for network communication (Zimmerman 1980), or can be a reference architecture that drives an architecture implementation, such as 4D/RCS to be described subsequently. Domain models can also be subdivided in terms of a technical domain, such as communications network, computation, agents, software, etc.; or a

system application domain, such as manufacturing, military weapon systems, business enterprise systems, etc. The focus here is on domain modeling that could be integral or otherwise aid in the development of an MIGVS concept architecture.

Classification and coding (Tatikonda and Wemmerlov 1992) is "a methodology which organizes entities into groups (classification)" with codes to "facilitate information retrieval" or to otherwise manage information about those entities. Relative to manufacturing, it is one way to implement "group technology" where the entities can be "parts, assemblies, process plans, tools, instructions, etc." Similar classification schema are used to organize domain "parts" for a variety of purposes. The PWBS portions of MIL-STD-881C essentially forms a component classification schema for all defense material systems that can be used for engineering, cost analysis and linked to 3D CAD. In similar fashion, automotive firms, such as Ford Motor Company's Corporate Product System Classification (CPSC) codes, use a classification schema to support their entire system life cycle (Riff 2010), to include aftermarket support, via their product life cycle management (PLM) system. When applied to a conceptual design phase, these types of schema can be considered to be a form of domain modeling that supports conceptualization of the system; a way to organize and structure the domain logic that includes a common language for that logic. These types of approaches apply better to the physical mechanical portions of the systems, less so to cyber components, particular those above control systems that can be directly related to the mechanical systems that they control.

As in concept design for product development, domain modeling in software engineering is a way to capture concepts of the problem domain as regards software. It can be applied to business process modeling for service oriented development, real-world objects in OOAD, or to any domain that the software is meant to address (Evans 2004). However, software domain modeling is more focused on software engineering techniques for model creation during a project or on some form of software product line engineering for an enterprise, not on the specific generation of a reusable application structure or on the software product line itself, though within a given company, there are apparently many examples of product line software (SEI 2016). They are primarily focused on

interoperability and composability of components to improve software design and development and less on early concept modeling and trade space support.

Fundamental techniques for domain classification can range from a simple taxonomy to a machine readable formal ontology. Taxonomies typically organize objects hierarchically in a tree structure and can have generalization/specialization and parent/child relationships and have provide useful for developing domain vocabularies and organizing domain entities from at least one aspect or view. Ontologies can include taxonomic relationships but also permit other types of relationships to address multiple aspects and have a greater emphasis on formal specification. Modeling complex engineered systems do not lend themselves to classification in a single hierarchy. In OOAD methods (Booch et al., 2007), "large application discrete systems" relative to the software only, have both "is part of" and "is type of" hierarchical relationships. The MIGVS domain with multiple forms of logic to include discrete software plus a physical dimension cannot be captured in a single hierarchy. It is possible that a MIGVS domain ontology (Semy, Pulvermacher, and Orbst 2004) could provide the necessary structure and semantics for a classification schema. An example of a domain ontology as a category of ontology is illustrated in Figure 8. The general idea is that an upper ontology can be used to define basic concepts about the world that can then be used to build a domain ontology, perhaps through one or more intermediate ontologies. Any given domain ontology ideally could be accessed by reasoning agents over the semantic web (Ding et al. 2005). Given the lack of standardization of upper ontologies and the difficulty in defining base concepts suitable for all domains, this approach at best is more notionally than formally linked, with the possible exception of biomedical domain (Smith et al., 2006).

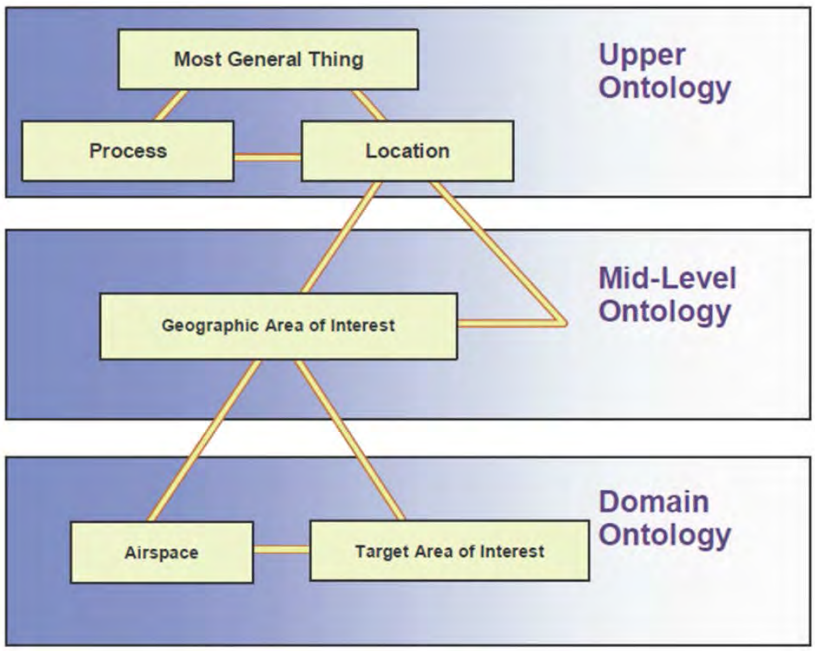

Figure 8. Domain Ontology. Source: Semy, Pulvermacher and Orbst (2004).

From a SE standpoint, some ontological focus has been placed on SE as a process for integration (van Ruijven 2013). An area of limited focus is the development of an ontology for systems. The DODAF Version 2.0.2 adapted an enterprise architecture ontology as the conceptual basis for its data metamodel (DM2) and architecture description. However, it is unlikely that these concepts have the appropriate basis for all DOD systems (Giachetti 2015) as opposed to just DOD enterprise systems or those that approximate enterprise systems. An MIGVS ontologically is composed of many "base" system concepts or classes (i.e., a cyber-physical system, a mobile system, a military system, a vehicle system). One would expect that each of these in turn have at least one unique concept that distinguishes it from other system types. Unfortunately, no such system intermediate ontology exists linked to a higher level ontology of underlying concepts fundamental to all systems.

Software domain concepts like OOAD, model views, separation of concerns, crosscutting concerns and constraints, and aspect-oriented programming, are essentially techniques to address the multiple types of concept entities and relationships embodied in relatively complex software. However, there are often underlying assumptions about the

type of software (e.g., discrete software systems), and about the type of underlying hardware (e.g., hardware that runs discrete systems software). Some of these approaches could be useful to systems like MIGVS, but would need to be extended to cover all forms of software and logic, all forms of computational hardware and all the physical mechanical hardware relative to the domain. Furthermore, to extend to MIGVS as a system application domain would require the use of reference models that can more explicitly capture the critical concepts for MIGVS as a domain. Each reference model captures critical concepts from a certain aspect, but all has appropriate crosscutting relationships to concepts in other reference models in the domain. These reference models could also be developed to at leverage fundamental concepts in the appropriate upper ontologies.

2. Model-Based System Architecture

A MBSE survey (Estefan 2008) provide a comprehensive capture and synopsis as well as key references relative to MBSE to include definition, advantages, leading methodologies and other related aspects. Most of the MBSE methodologies have either an explicit approach to defining a system architecture or an implicit one driven by the methodology. They will be reviewed as a body of work focused on system architecture and not as a comprehensive review, comparison, and critique of each specific methodology. The order of the review is arbitrary and aimed at capturing a superset of all critical concepts and features. Duplicative features found in subsequent methodologies are not repeated. The superset of concepts and features will then be compared and contrasted with the specific objectives and concepts of this research.

a. INCOSE Object Oriented Systems Engineering Methodology (OOSEM)

Both a general overview and an application of OOSEM with a focus on architecture are shown in Figure 9. Key concepts are:

(1) Compatibility with object oriented analysis and design (OOAD)— though a full set of reasons why this compatibility is desired nor the original objectives of OOAD are not articulated, many are stated (OMG MBSE Wiki 2011) or can be inferred: use case/scenario analysis to establish measures of effectiveness (MOEs), requirements elaboration

and measures of performance (MOPs); the concepts of classes, inheritance and object models for logical decomposition and white box/black box elaboration, and integration with object oriented software.

(2) Use of general purpose modeling language, particularly UML/SysML—the advantages of a general purpose modeling language such as SysML over a niche language is its potential to capture a wider market and therefore offer a more expansive ecosystem (trained users, compatible software, tool vendors, etc.) at a lower cost. A general purpose language will also be easily understood by not only systems engineers, but other engineering disciplines that must work or integrate with a system model.

(3) Multi-Levels of architecture abstraction—the 2nd pyramid view (House and Pearce 2012) in Figure 9 shows multiple view into an architecture that progress through an OOSEM process. In addition to the use cases, it introduces a functional architecture that can be expressed in non-object oriented techniques such as functional flow block diagrams (FFBDs), a technology and implementation independent logical architecture, and finally a physical architecture that can be arrived through a synthesis and trade process.

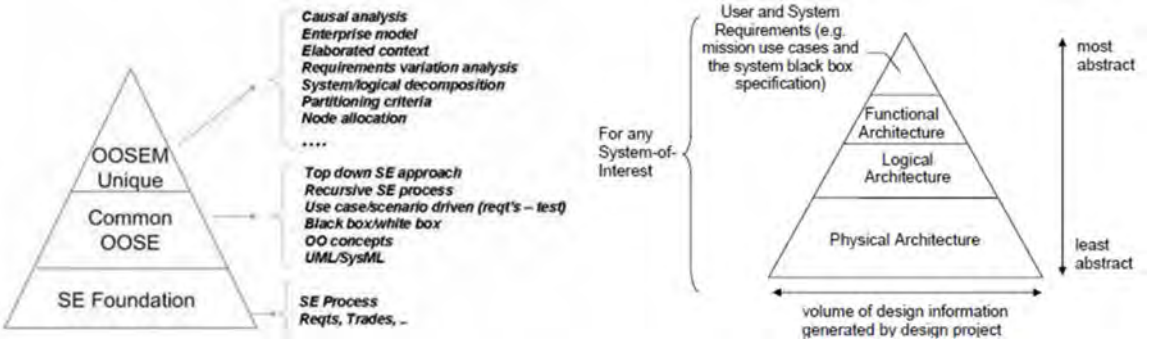

Figure 9. INCOSE OOSEM. Source: Estefan (2008) and House and Pearce (2012).

b. IBM Rational Rhapsody and Rational Harmony for SE

Since this methodology (Hoffman 2011) is focused on best practices that use UML/SysML, it does not differ in principle from INCOSE OOSEM and those objectives. However, there are few differences and many things that come to light in this far more detailed methodology:

(1) Integration with real time embedded components—rather than the more general objective of integration with software, this methodology has a strong focus on embedded components or what it terms as systems that

are highly state based. This leads to much more reliance on SysML statechart diagrams.

(2) Behavior diagrams applied to use cases—all three SysML behavior diagrams: activity, sequence and statecharts are considered necessary because of their different strengths to fully elaborate the internal functions and behaviors required of the architecture

(3) Use cases for "logical decomposition" - decompose functions and operations using multiple behavior views, use cases baseline system requirements and provide behavior (and therefore also function) allocation to architecture components.

(4) Design concept and design synthesis—The design concept model identified the major physical components. Design synthesis adds weighting/criteria to the model to evaluate major alternatives. Once major components are defined, detailed design defines ports/interfaces and to state behavior to those components.

(5) Architecture visualization and verification—use the model description for visual verification, animation, automatic generation of sequence diagrams.

(6) Conceptualization Modeling—in RUP terms this would be part of inception and elaboration as well as iterative development. Key idea being that a whole model of the system is created relatively quickly in the development and then refined and elaborated toward a design.

(7) Separation of Concerns/Software Frameworks—though often implied or sometimes even explicitly referred to in RUP methodology and associated concepts, separation of concerns deserves its own treatment since it leads to the concept of views (Goedicke 1990) in software frameworks and the ability to "neglect" certain parts of information for the sake of other parts of concern to a particular stakeholder or set of stakeholders. This in turn promotes the concept of modularity and linking the idea of a main function for a given modules.

c. Jet Propulsion Laboratory (JPL) State Analysis

The JPL state analysis MBSE approach is based on a "state based control architecture" (Wagner et al. 2012) shown in Figure 10. It is perhaps more suited for a domain, or at a point in the systems engineering process, where there is more of a priori or clear distinction between the behavior system and the overall system topology/constraints and physical mechanical systems, and where there is a less complex

external environment than that found in the MIGVS domain. However, in terms of early or concept design MBSE and behavior modeling, and as a potential companion methodology to be used alter in the system design, there are a few key concepts that should be considered:

(1) Systems and software engineering interdependence modeling for complex systems—as systems have become more complex, reliance on functional based methods and narrative requirements are no longer sufficient between system and software design. A model of the behavior of the system needs to be defined by systems engineering and used for software design.

(2) Complex or multiple interacting control systems—the domain of interest is not simply real time or closed loop control, but multiple interacting control systems that need to adapt to goals or user input, uncertainty and faults.

(3) Goal based behavior is integrated into the model—Goals are integrated into the model as state intent that reflect what the operator or agent would like do as a way of integrating the operator as part of the control system. Complex activity can be planned and executed via goal networks or re-planned based on goal changes and faults.

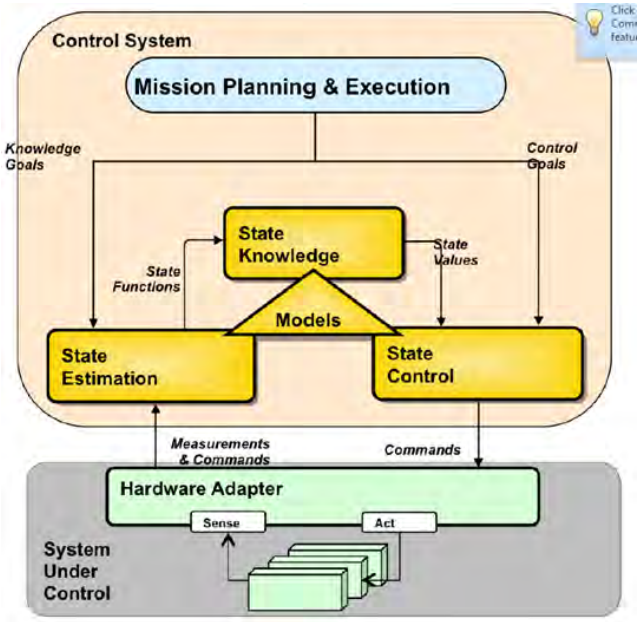

Figure 10. State Based Control Architecture. Source: Wagner et al. (2012).

d. Vitech Model Based Systems Engineering (MBSE) Methodology

The Vitech MBSE methodology (Long and Scott 2011) adds methodology, system model, and modeling language features that have not previously been highlighted:

(1) Multiple layers of system design abstraction—Vitech has its own detailed approach to this called STRATA™, but as a general feature it can be summarized as follows: the entire system design is captured at a level of abstraction, the abstraction is refined and detailed in stages until system design is complete, each stage must satisfy completion criteria before proceeding to the next stage. The culmination of each stage results in system verification and validation for that stage of abstraction.

(2) Model the whole system—this requires having a complete model in terms of depth and breadth, behavior (time independent and dependent) and structure, and boundary conditions. It is more than a sum of views but an integrated whole at successive levels of detail as the design progresses.

(3) Model system context—a system's functions and interfaces cannot be fully understood without understanding their interactions with the external environment. This in turn requires a specific focus and explicit rationalization of the system boundary.

e. Dori Object-Process Methodology (OPM)

The "integrated process and object" approach in OPM (Dori 2014) defines a formal relationship between functions/processes and objects into a single diagram to promote more integrated reasoning that requires less generation of diagram types compared to multiple views. It creates bridge between structured methods with object-oriented methods. Dynamic behavior of objects are reflected by an object state change driven by processes.

f. Integrated Systems Engineering (ISE) and Process Pipelines in Object Oriented Architecture (PPOOA)

The ISE PPOOA (Sanchez 2012) approach combines "classical" engineering approaches with model based approaches, particularly at the systems engineering level. However, it does add some unique concepts for a more complete model based approach not covered yet.

(1) Operational Concept—this is an abstract model of the operations of a system. A scenario than is a particular path through this concept for a given set of conditions

(2) Capability—the system's ability to perform an effect. Scenarios then are transformed into a set of system capabilities

(3) Quality Attributes/Constraints—constrain the system architecture relative to meeting its functions or performing its capabilities

(4) Early concurrency modeling—for software intensive mechatronic based systems it is important not to just establish the logical relationships and collaboration in the object model, but to model the concurrency to drive out timing constraints.

Taken as a whole, the MBSE methodologies reviewed provide a superset of concepts that need to be considered in the development of a model based MIGVS concept architecture. However, even taken as whole, they do not provide the concepts and features necessary to meet the objectives of this research. Particular areas of note are as follows:

a. Object oriented modeling of a system—As noted in Figure 9, the OOSEM methodology views the "logical architecture" as an abstraction of the "physical architecture." This is a different view of the relationship between logical and physical from software OOAD (Booch et al. 2007) and the software "4+1" model (Kruchten 2004). The logical view enables a relatively granular view into software components based on how they interact, not how they are assembled. The equivalent of software physical assembly of those same components is provided in another view, referred to as the physical view in Figure 3. As such, they provide different views or abstractions of the same "physical components," but the logical view is not an abstraction of the physical view or deployment view. Several of the other methodologies do not as explicitly model a logical architecture, but seem to infer a similar view. OPM and ISE POOPA seek to combine structured and object oriented approaches, with the former citing integrated reasoning of process and objects, but that integrated reasoning can be provided by having two views into the same components as practiced in the aforementioned software approaches. The objects need to encompass more than software however, the objects

must also consist of computational hardware, physical-mechanical components, sensors, etc.

b. Assessing attributes that impact behavior—A focus of ISE POOPA is quality attributes or constraints and early concurrency modeling for mechatronic-based systems, which can be considered a cyber-physical system or least a pre-cursor thereof. These attributes from a system standpoint can be considered in two varieties: physical attributes and behavior attributes. The physical attributes, such as weight and volume, are best addressed in physical views of the architecture. Here again, as practiced in software OOAD, there is an object-oriented view that can be used to address behavior attributes such as timing, often referred to as the "process view." The executable view model provides that view at a system level of abstraction. The overarching multi-view model provides integrated reasoning of both behavior and physical attributes through different views of the same components. It is not clear how such reasoning would take place with any of the other MBSE approaches.

c. Use cases and intelligent operation—The concept of a use case and a user leads to some confusion as a system acquires more endemic intelligence. This is best illustrated for a system facilitating a business transaction, such as an automatic teller machine (ATM). If the ATM was only partially automated and still required a teller to help it function, the teller is using the machine to facilitate the customer's objective. In that sense the teller is operating and the customer is using. The teller is required because their intelligent capability was not fully incorporated into the system design. To consider the teller as an actor in a use case distinct from the system and as an a priori consideration in the design, is to arbitrarily limit the design and confuse the purpose of the system which is to provide a transaction for a customer. To not model the teller's behavior in any way and commit to full automation is to arbitrarily limit the system alternatives and to lose valuable insight into needed behavior. The use case of a customer is relatively enduring and distinct from the cases involved in the operation and maintenance of the system, which should be trade space and concept design dependent.

d. Intelligent behavior modeling—none of the MBSE methodologies explicitly model intelligent operation behavior. The JPL SA does provide a mechanisms

for goals, but they seem to be human provided and managed, not a way for the system to operate intelligently. The focus of several methodologies seems to be more on a control system and the thing being controlled, not on intelligently operating to goals and interpreting the external environment as required. The concept of an agent enables both intelligent behavior modeling and interaction of with control system objects which can then interact with physical-mechanical systems. The operator can be modeled as part of the system and the "customer" modeled external to the system or as part of the context. For a MIGVS, the "customer" is not always a willing participant in the transaction (e.g., an enemy vehicle hit with a large caliber round). The effects or goals can reflect a change of state in the external environment. The external environment or context and its interaction with the system's intelligent behavior, brings an increased focus and emphasis on modeling that context. The current MBSE methodologies do not explicitly model intelligent behavior and its interaction with context modeling.

3. Architecture and Modeling Design Languages

MBSE/MBE approaches require that system design and system conceptualization should be supported by models. As indicated previously, 3D CAD models support the physical architecture conceptualization and embodiment design, with focus on geometric and some other physical attributes and relationships. As a descriptive design language, 3D CAD provides little support to understand and capture the attributes and relationships of objects as they impact behavior. These attributes and relationships require a different type of design language, yet can still integrate with 3D CAD for overall system conceptualization and embodiment design. Three such design languages are the Object Management Group's System Modeling Language (SysML™), the Society of Automotive Engineers (SAE) Architecture and Analysis Design Language (AADL), and the Modelica Association's Modelica®. These design languages will be reviewed only for how their most basic underlying architecture concepts of structure, behavior and interconnection, support MIGVS conceptualization and embodiment design, not on their overall efficacy to support system engineering and product development.

All three provide constructs for representing behavior, structure and interconnection between structural elements. SysML, as an extension of the Unified Modeling Language (UML), provide a form of object oriented structure called blocks that can have “is a type of” and “is a part of” relationships. Native property types supported in the language are biased toward viewing the blocks as hardware, but anything in SysML can be typed as a block, including the interconnection mechanisms. It provides high level behavior description in terms of use case, activity and sequence diagrams. Modelica (Fritzson 2012), also provides an object oriented, structure, with a focus toward hierarchical physical decomposition covering multiple domains (e.g., electrical, mechanical, hydraulic). The interface classes are also focused on these physical domains via types of energy. Critical to the approach is to link the structure to mathematical equations at the leaf level for execution. Though it provides support to event or discrete time, beyond control systems, it does not attempt to address high level behavior and software structure. It does have a UML profile called ModelicaML that allows Modelica constructs to be represented in UML/SysML.

AADL, as shown in Figure 11 (Hudak and Feiler 2007), is also focused on the “lower end” of the system architecture, but focused more extensively on the computational architecture and behavior of the system. As such, it models both the software and computation in connection with physical system components at a level of abstraction above design. Going counterclockwise from the upper right of Figure 11, its view of system architecture is one of a set of control systems interacting with a set of physical devices. Each control system can be further decomposed and the behavior of the software in its computation execution can be modeled and examined. The ports shown in Figure 11 are actually groups of logical interfaces that can be elaborated and/or decomposed. Software can be modeled as threads and latency examined through the processing, memory and bus interconnects. Though there is the ability to model software, computation hardware, and physical mechanical components, like Modelica there is no facility to address higher levels of behavior such as those that might embodied in discrete software.

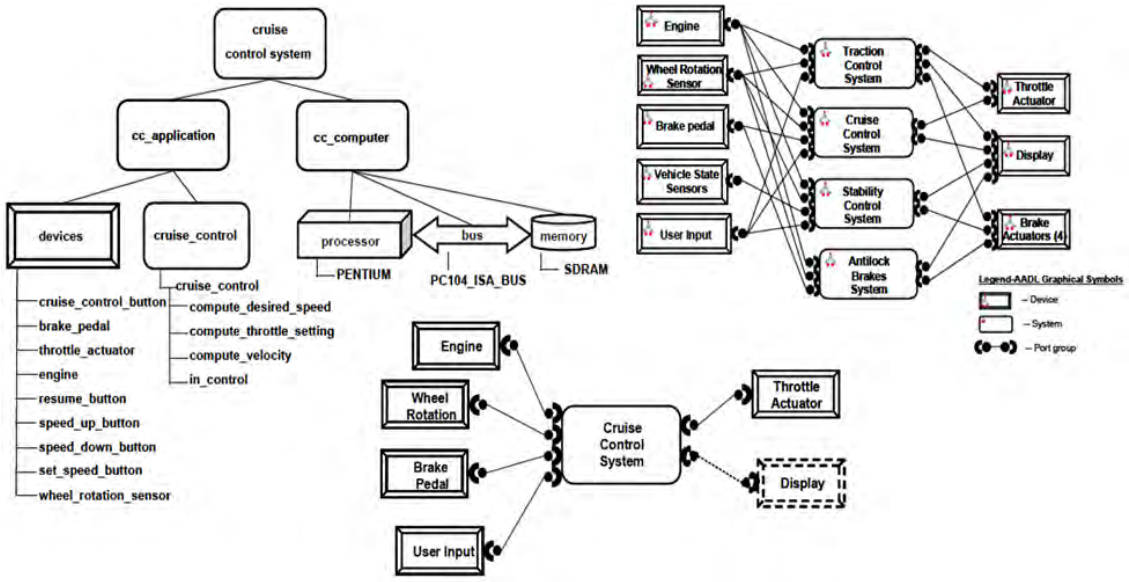

Figure 11. AADL Cruise Control System Hierarchy. Source: Hudak and Feiler (2007).

Similar to Modelica, there has been some work to look at integration of AADL with SysML (Behjati et al. 2011). There is also effort at improving the link (Espinoza et al. 2009) between SysML and a UML 2 Profile known as Modeling and Analysis Real-Time and Embedded Systems (MARTE). This along with the Modelica integration work identified above indicates that SysML is best suited at the higher levels of abstraction and at higher levels of a behavior hierarchy. AADL and Modelica then are perhaps better suited at lower levels of abstraction and of the behavior hierarchy. This in turn suggests that at the level of system concept design, SysML would be better suited to capturing and analyzing the system architecture relative to major trade space analysis. Once the computation architecture and control elements are defined, the detailing of the interfaces for the embodiment design as well as preliminary design of the configuration items, might be better supported by a variety of languages, such as AADL or Modelica and perhaps other design languages, such as those for software design. Alternatively, depending on project needs, the SysML model could be further elaborated to achieve the same objective and then AADL or Modelica employed at the level of product architecture and design. Regardless, for SysML to integrate with lower level architecture or design models, it must provide a suitable overlap of information. This reinforces the need to model both high level behavior with lower level real time and physical mechanical

behavior as interacting objects at a system level of abstraction. If this capability is not built into native modeling language, it must be created a specialization of more general methods.

B. CPS SYSTEM ARCHITECTURE

A MIGVS has been evolving toward being an overall CPS (i.e., an increasingly level of computational control over its physical processes). A fully autonomous ground vehicle would in fact need a base CPS layer in order to realize the higher levels of intelligent behavior required for autonomy. As such, system architecture and modeling approaches for systems that are "CPS-like" will need to be reviewed in addition to those approaches for systems that are explicitly identified as CPS.

1. "CPS-Like" Engineering, Architecture and Modeling

The emergence of CPS systems did not wait for the term cyber-physical to be defined. Several types of systems have been described by terms that are essentially CPS or have significant overlap with CPS features. These include systems described as intelligent, mechatronic, agent based, multi-agent, heterogeneous, autonomous and robotic. Systems are also described as combinations of these terms. Increasingly, emerging research in similar areas are being described as CPS or at least as relating to a particular problem confronted by CPS. Research in these areas that are most relevant to the MIGVS architecture conceptualization will be reviewed.

The 4D/Real-time Control System (RCS) reference model architecture (Albus and Meyestel 2001) models a CPS in general and an MIGVS in particular. The 4D refers to the four dimensions of intelligence defined as Sensory Processing, World Modeling, Value Judgment and Behavior Generation that are embodied within an RCS computational node (Albus 2002) as shown in Figure 12. These nodes are arranged in a hierarchical control structure that can represent a vehicle system and military unit as shown in Figure 13. Additionally, there is a companion methodology for capturing knowledge and representing it within 4D/RCS shown in Figure 14. The 4D/RCS approach addresses many important concepts for a MIGVS which are briefly summarized as follows:

(1) Multiple forms of behavior logic are captured. It include basic feedback control, intelligent control, and task-oriented behavior.

(2) Integration of computational or cyber behavior with physical mechanisms. Systems can be formed by scaling up from the control of physical systems to the highest level mission behavior and include timing constraints.

(3) Repeatable methodology for assessing the computational needs. Operational tasks can be captured from basic doctrine and related to computational nodes and then mapped to system behaviors.

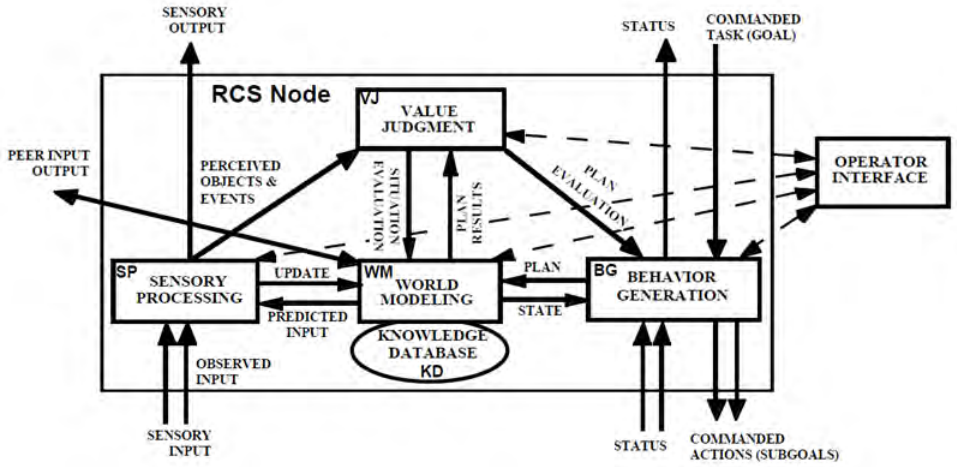

Figure 12. 4D/RCS Computational Node. Source: Albus (2002).

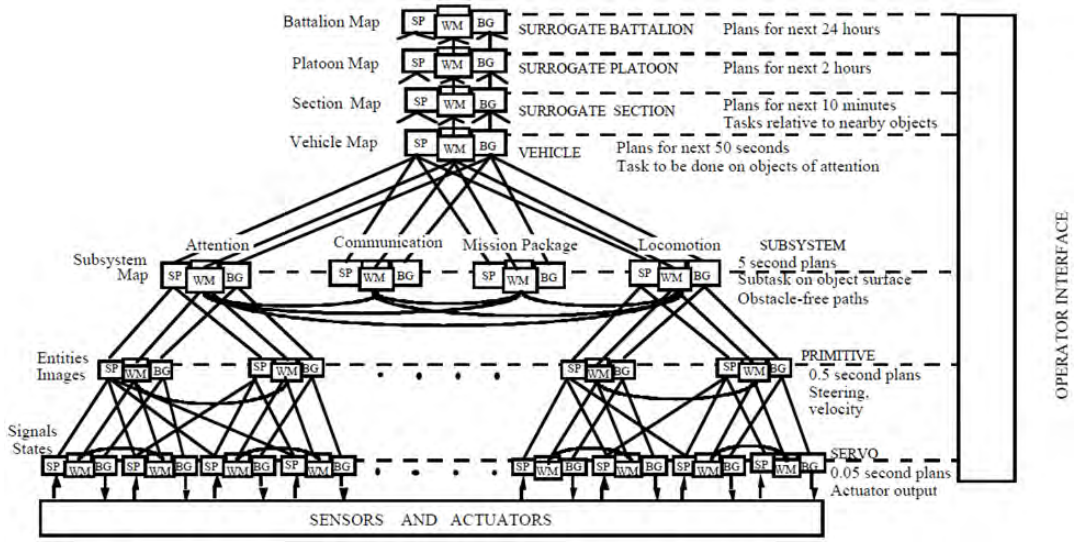

Figure 13. RCS Based Notional Military System and Unit Structure. Source: Albus (2002).

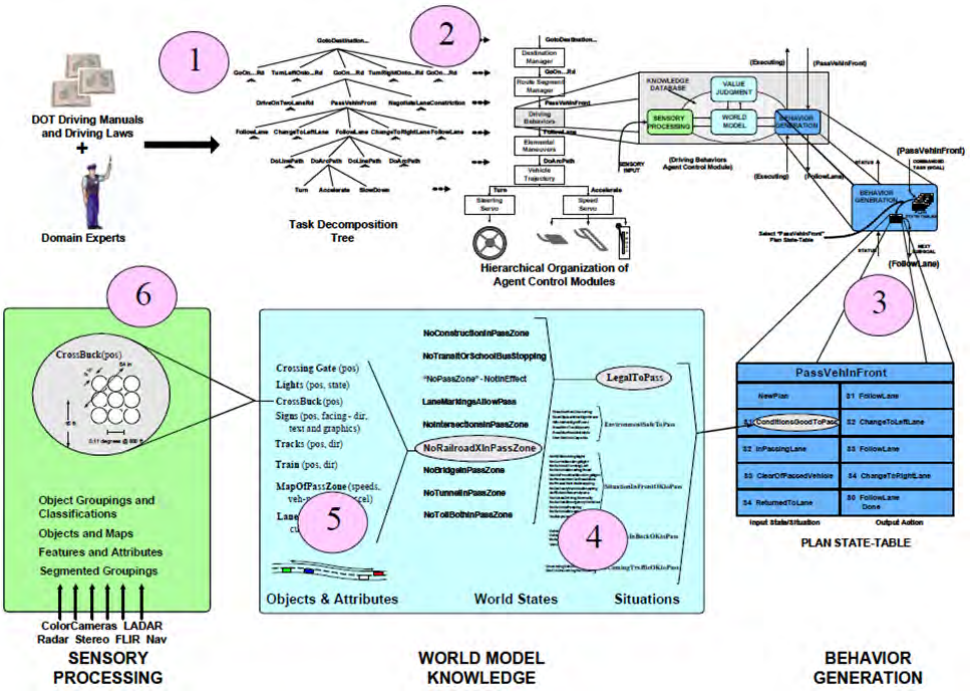

Figure 14. RCS Methodology for Knowledge Capture and Representation. Source: Albus and Barbera (2004).

As a reference model architecture, 4D/RCS is a blueprint for the computational architecture design. It does not have the scope or the necessary abstractions to conceptualize an MIGVS system architecture. The overarching system is not addressed so it does not enable the necessary trade space to determine the optimum system concept in terms of cost and operational effectiveness. Similarly to some of the MBSA architectures within the MBSE methodologies and the architecture design languages, the 4D/RCS reference model architecture could be applied to system embodiment design of the computational architecture once the system has been conceptualized and major configuration items identified, particular if the concept decision identified an autonomous system solution, determined why an autonomous solution achieved optimum mission effectiveness, and determined that a 4D/RCS based implementation was the most effect approach for realizing the systems computational architecture.

4D/RCS can also be described as an agent based or multi-agent system reference model architecture. Each RCS computational node represents a certain type of agent. In software agent terms (Mayk and Regli 2006), multi-agent systems (MAS) “incorporate

several agents where the goals of the agent system are achieved through the interaction of the individual agents," and a MAS reference architecture is one of many possible derivations from an MAS agent reference model and can guide the development of many MAS designs. Depending on the level of abstraction, agents can be considered almost identical to objects or quite distinct. However, even at the low abstraction level of software programming, agent oriented programming (AOP) can be defined to be a "specialization" (Shoham 1993) of object oriented programming (OOP). At the level of software design language the differences are less pronounced. Agent modeling (Bergenti and Poggi 2000) and interactions (Regli et al. 2014) have been introduced into UML for example. At the even higher abstraction level of concept architecture, differences between agents and other objects should be even less pronounced

Beyond agent-oriented software engineering and agent-oriented programming languages alluded to above, there are also high level agent-based methodologies (Tveit 2001). Of particular interest is Multiagent Systems Engineering (MaSE) (Deloach et al. 2001) with its link to architecture and "system design" as shown in Figure 15. Of particular note is its use of goals in lieu of or at least one type of requirements and the use of roles to structure tasks and "agent classes." The latter is useful to mitigate differences in agent objects and other "system objects." As shown in the review of architecture design languages and in OOAD, objects can be a physical-mechanical mechanism, a computer, a control system, a segment of code, etc. They all represent some physical or "real-world thing" if we adopt the OOAD view that software has a "physical" realization in code, programs, executables, assemblies, etc. For an "agent object," its physical realization can be software code and perhaps sensors and actuators, but it can also be a human operator when the entire system is considered instead of the software only. More definitively, it can be an "instance" of a given human operator performing a role. Whether assigned to software and machine or to a "human instance," agents cooperated together to achieve goals, which is a different and necessary augment to the more standard view of functions and requirements.

"An intelligent mechatronic system is capable of achieving given goals under conditions of uncertainty" (Rzevski 2003), as compared to an automated mechatronic

system which self-regulates to predictable changes in the environment. As such, the mechatronic system domain seeks to leverage architecture techniques already discussed like multi-agents, architecture design languages etc. However, it does bring a unique perspective on system concept design as a necessary first step to link to multiple design disciplines and integration with MBSE. A specific approach to model based (Thramboulidis 2010) mechatronic system concept design is shown in Figure 16. Similar to AADL, it defines mechatronic components (MTCs) that encompass electronics, software and mechanics. These component areas with their specific domain model are then integrated to a system view to form a "SysML 3+1 model." System concepts are then iterated with component concepts and technology selection until a final system architecture is selected. This approach does not address higher level logic and trades relative to human task/agent components. Also, though it provides component linkages to product tools, it is not clear there is enough fidelity and scope at the system logical structure and behavior to fully assess the interdependence between MTCs and address all MIGVS component types.

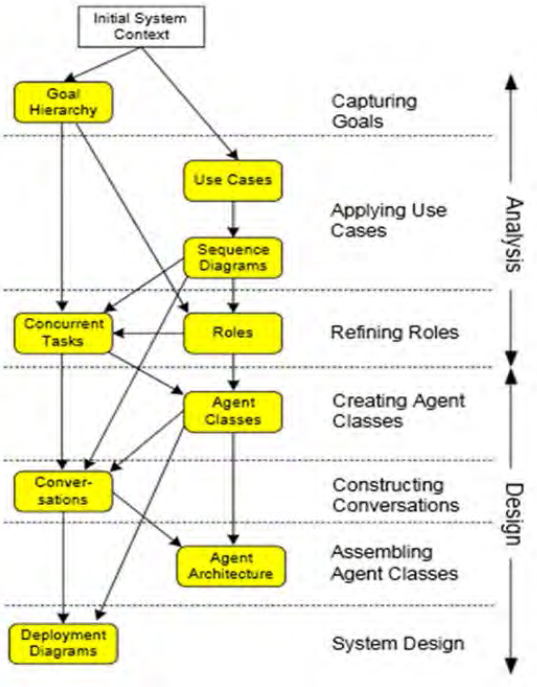

Figure 15. Multiagent Systems Engineering. Source: Deloach et al. (2001).

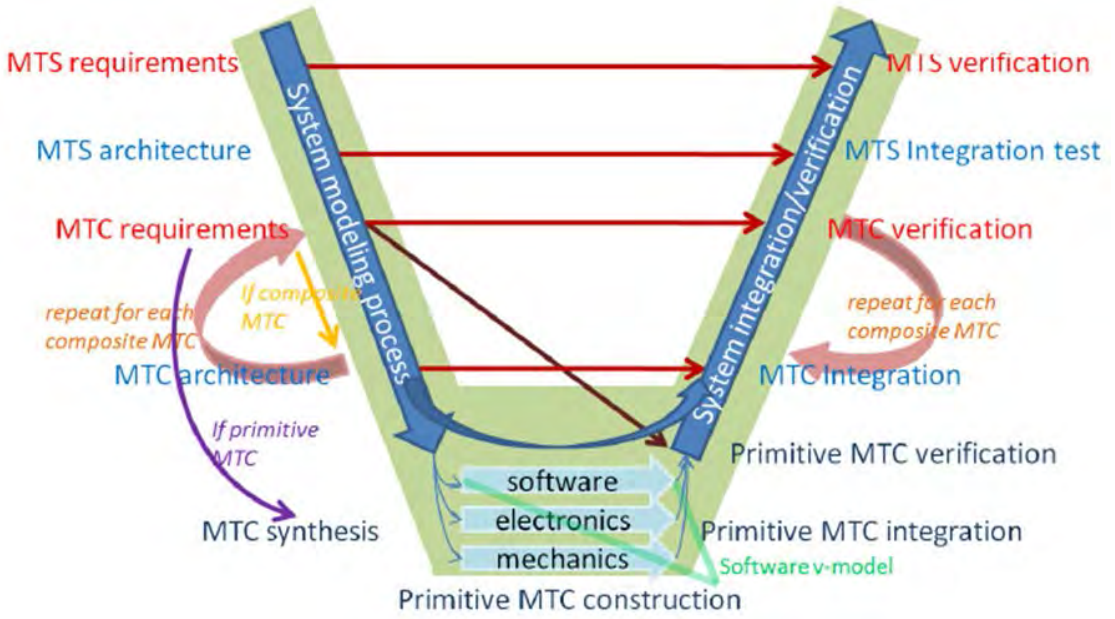

Figure 16. Mechatronic System V-Model. Source: Thramboulidis (2010).

Intelligent manufacturing systems is a domain that has seen a certain evolution and variety of techniques in intelligent automation with some recent emergence connecting in CPS. A lineage has been identified (Leitao, Marik, and Vrba 2013) linking holonic manufacturing systems (HMS) and architectures form the international intelligent systems manufacturing (IMS) program, International Electrotechnical Commission (IEC) 61499 for distributed control, multi-agent systems and standards, service-oriented agents, and holonic agents. A holon in this context has been defined (Van Brussel et al. 1998) to be an autonomous and cooperating building block of a manufacturing system" where holons can be considered whole or part of another holon. Holons "cooperate to achieve a goal or objective" within a holarchy that limits their autonomy with cooperative rules. A view of holons and their structure from the PROSA HMS architecture is shown in Figure 17 using UML notation for specialization and aggregation. An HMS is composed of three basic holons: order, product and resource. These in turn can each be specialized into many types as shown for "Resource Holon." The concept of holons naturally lends itself to domain modeling, object oriented and multi-agent system concepts. However, whereas agents have been successfully deployed in "production planning, scheduling and logistics" they have not been successfully deployed in factory automation beyond the

laboratory or prototypes. However, again, industrial multi-agent systems are still viewed as potentially promising.

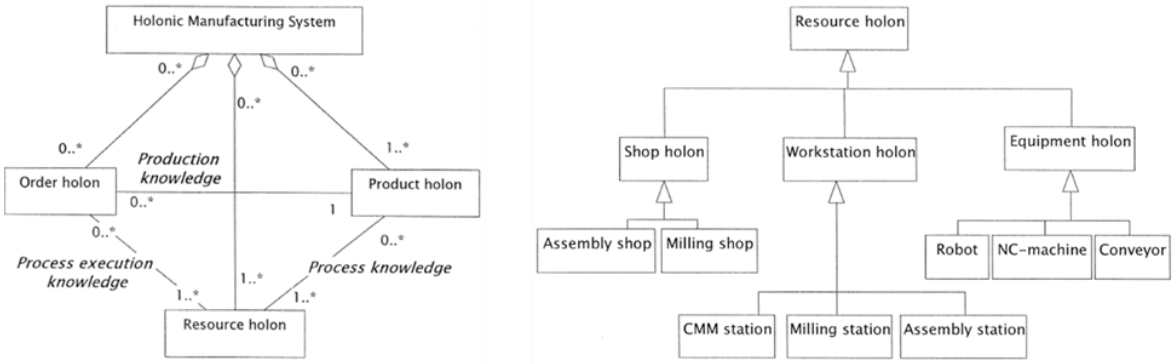

Figure 17. Holonic Building Blocks in Manufacturing Systems. Source: Van Brussel et al. (1998).

The lineage from HMS/IMS to IEC 61499 has met with more factory automation success as well as direct CPS linkage. IEC 61499 is meant to define a standard for high level control or "distributed intelligence" (Cruz Salazar and Rojas Alvarado 2014) above the programmable logic control layer and first emerged as a lower level extension of holonic and agent based approaches. The key construct is a functional block (FB) as shown in Figure 18. All functional blocks share a flow of events and a flow of data. There are basic FBs for general behavior, service interface FBs for network or environment interface, and composite FBs that enable composition and scaling of a complete system. This approach enables a hardware independent event driven execution of lower level control systems. It has also been viewed as competing (Kruger and Basson 2013) with a MAS approach or as a better way of realizing (Sorouri et al. 2015) agent like behavior at the device level. There also seems to be few practical realizations of the overall IEC 61499 standard. The simplicity of FBs may lead to a lack of flexibility in intra-communication for complex systems or to adapt to changes.

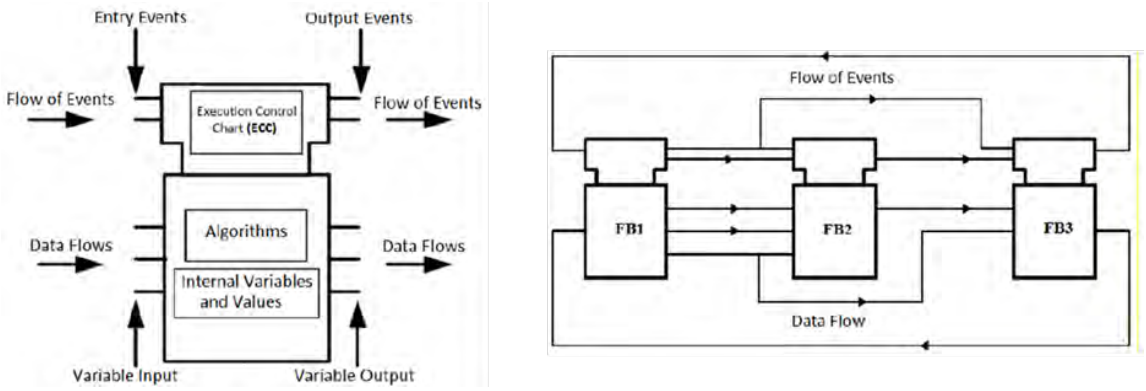

Figure 18. IEC 61499 Functional Blocks. Source: Salazar and Alvarado (2014).

The FBs facility to link discrete events to continuous time control has led to some initial investigation of model integration (Pang et al. 2015) into the Ptolemy II framework (Ptolemy II 2014), to include introducing time stamps to FB semantics (Vyatkin et al. 2015), as a means of "maintaining cyber-physical system properties." A higher level architecture or "unified system framework" (Lee, Bagheri and Kao 2014) for manufacturing systems has been offered that incorporates a cyber-physical layer as one of 5C shown in Figure 19. The CPS layer is seen as a "central information hub" and a means to harness big data enabling high level resiliency goals. This general framework would need some implementation detail before it could considered an architecture and it is not clear why cognition is considered outside of CPS versus a certain model of computation. Though lessons can be learned for the MIGVS domain, the dynamic environment and system constraints of mobility will make manufacturing CPS solutions difficult to apply directly.

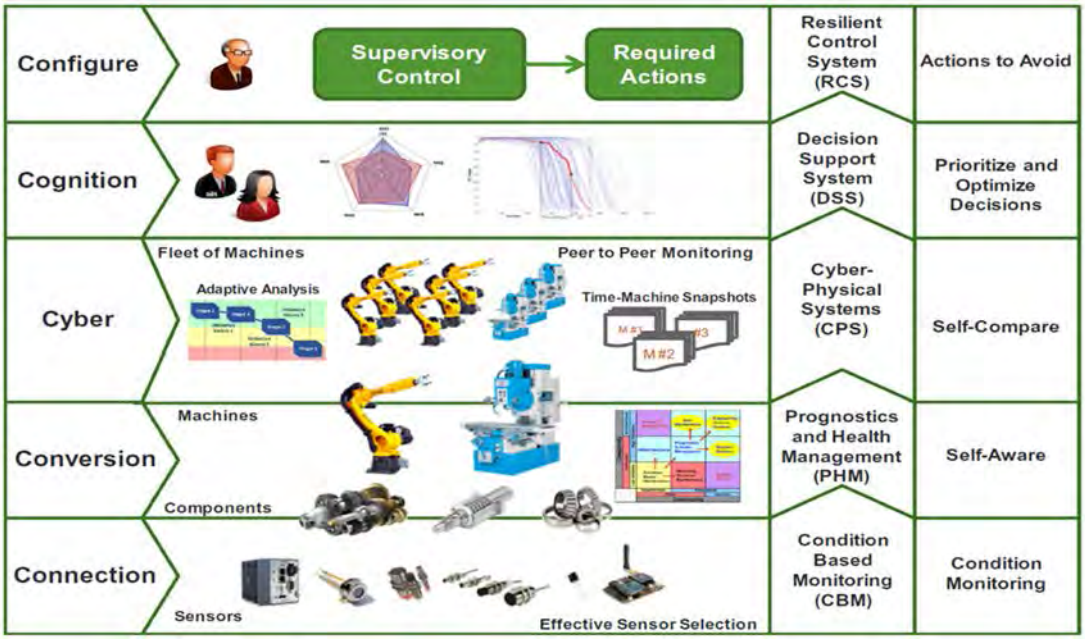

Figure 19. Manufacturing System 5C Architecture.
Source: Lee, Bagheri, and Kao (2014).

2. CPS Engineering, Architecture and Modeling

The morphology of terms in intelligent manufacturing is a clear example that the concerns of CPS have been around for some time now, at least for certain domains. For MIGVS or intelligent manufacturing systems, CPS emergence has resulted in an increased recognition of certain concerns along with new approaches and techniques for addressing them. There is a greater recognition of the interdependence of multiple technical domains within computational and engineering disciplines as required to establish a foundation for CPS. These technical domains include (Baheti and Gill 2011): networking, control, software, human interactions, mechanical, electrical and others. A CPS survey (Khaitan and McCalley 2015) classified the crosscutting issues into design, aspects and applications. Another survey (Gunes et al. 2014) referred to challenges that addressed similar issues to aspects. The term quality attributes will be utilized to address these similar issues.

Cyber-physical systems and the Internet of Things (IoT) are sometimes used interchangeably. In this research, IoT will be considered as a type or domain of CPS or as

an enabler to a type of CPS. The IoT set of concerns overlap with a MIGVS, but each have their unique concerns. The unique IoT set of concerns is defined as distributed computing utilizing a web or Internet-like protocol, geographic dispersion of devices, composability, and large data volumes. A MIGVS is a system that contains a hierarchy of networked devices engineered to meet its capabilities within mobility and other constraints. Once engineered, it may also then be linked into a larger network with an Internet-like protocol as a "thing" within an IoT or an "Internet of systems." It may also expose its devices into other IoT type linkage, such as for maintenance. In this sense, the MIGVS is both a CPS and a thing that can participate in a larger CPS, as well as a set of things that can participate in multiple other CPS systems. Key distinctions are an internal system hierarchy and level of dynamic composition possible. CPS literature that address IoT concerns will only be reviewed to the extent it can be applied to MIGVS concerns.

a. CPS Architecture and Modeling

CPS design (Khaitan and McCalley 2015) was defined to include several areas. The particular area relevant to this research is "architecture and modeling." Khaitan and McCalley further subdivided this area into "model-driven development," "meta-architecture and meta-programming," "semantics," and "co-design." Properties will be discussed as part of attributes. Many of these areas overlap with each other and with considerations of attributes and domain applications. Semantics can include both behavior and properties. Models and behavior semantics can be intertwined. In addition to the IoT and MIGVS distinction, a distinction between concept design and embodiment or detailed design will further restrict the literature search. Consideration will be given embodiment or detailed design research to the extent that it helps determine what should be captured in an informative concept design.

Much of the model-driven development survey (Khaitan and McCalley 2015) included research and development focused on the concerns of IoT. One architecture example (Tan, Varun and Goddard 2009) as shown in Figure 20, highlights both the distinctions and the similarities between IoT and MIGVS. Similar to 4D/RCS (Albus and Meyestel 2001), it has an optional direct link of sensors or sensor processing, an optional

direct link of actor "motes" and control system or behavior generation, and an integrated model of the external environment or world. Unlike 4D/RCS, there is a relatively flat sensor and control hierarchy, a network and data base server distribution schema, and a human operator performing some undefined role. Like 4D/RCS, it classifies events relative to the hierarchy and gives them both spatial and temporal properties. However, because of its relatively flat hierarchy, it processes events only as symbolic information and does not include the iconic information at the layers immediately above the control system in 4D/RCS.

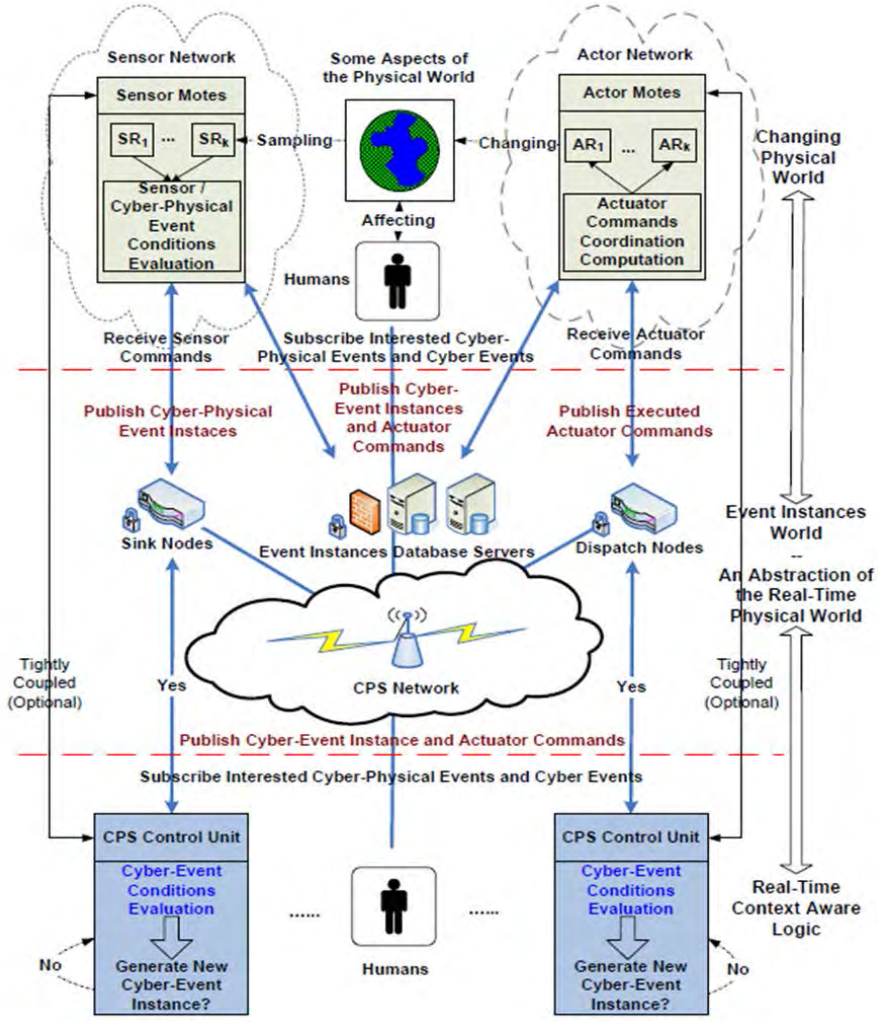

Figure 20. "IoT-like" CPS Architecture. Source: Tan, Varun and Goddard (2009).

CPS modeling challenges (Derler, Lee, and Sangiovanni-Vincentelli 2011) can also be distinguished by the "joint dynamics of computers, software, networks, and physical processes." The fundamental modeling challenge is combining the "sequential" nature and discrete event based software with continuous physical processes that can occur concurrently over time or during execution. A model based design methodology for CPS (Jensen, Chang and Lee 2011) has been proffered that makes extensive use of simulation for both analysis and verification. This puts the modeling emphasis on generating execution models versus descriptive or notational models. This methodology rests on two fundamental concepts: *platform-based design* (PBD) and *actor-oriented design*.

PBD is utilized to "separate application logic and architecture-specific software into modular components" (Jensen, Chang and Lee 2011). A platform is defined (Sangiovanni-Vincentelli 2008) as "a library of components that can be assembled to generate a design." As illustrated in Figure 21, a system platform has an application or functional space that consists of a set of designs that can be analyzed from the top down to select an instance. There is also an architectural space with a stack of platforms each of which consists of a set of designs and abstraction isolation between layers of the stack. Components in the architecture stack can be selected bottoms up based on rules. The mapping of the application to the architecture occurs via an Application Program Interface. One goal of this approach is understand and distinguish the functionality from the hardware implementation to achieve greater reuse.

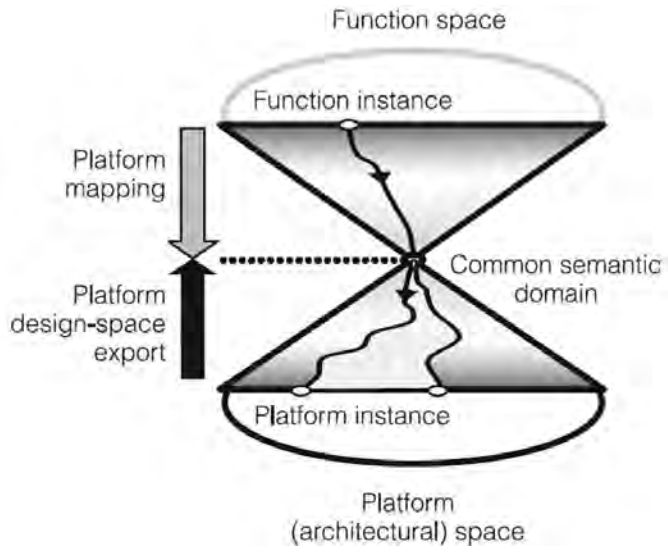

Figure 21. Platform-Based Design. Source: Sangiovanni-Vincentelli (2008).

This approach has been adopted and adapted for actor-oriented design of cyber-physical systems as shown in Figure 22 with an illustration of an architecture platform stack. A key problem for CPS is that the abstraction layers have "failed" (Lee 2008) to sufficiently isolate and/or express key attributes important to CPS through the layers. These attributes include predictability and reliability, particularly as regards timing properties. As Lee goes on to explain, at the chip platform components are produced that are reliable and predictable. However, the reliability and predictability are not expressed to upper layers and each layer increasingly introduces some level of loss. Additionally, CPS by their nature are concurrent (Lee 2008), which is another system attribute insufficiently addressed via software threading and hardware interrupts managed by the operating system. Solutions (Lee 2008) to these challenges can occur from the bottom up, but would require a significant modification to computer architecture and software practice, or can occur from the top down where "programs" are replaced by "models" of the system behavior and software "synthesized" from those models. The latter is accomplished via actor-oriented models.

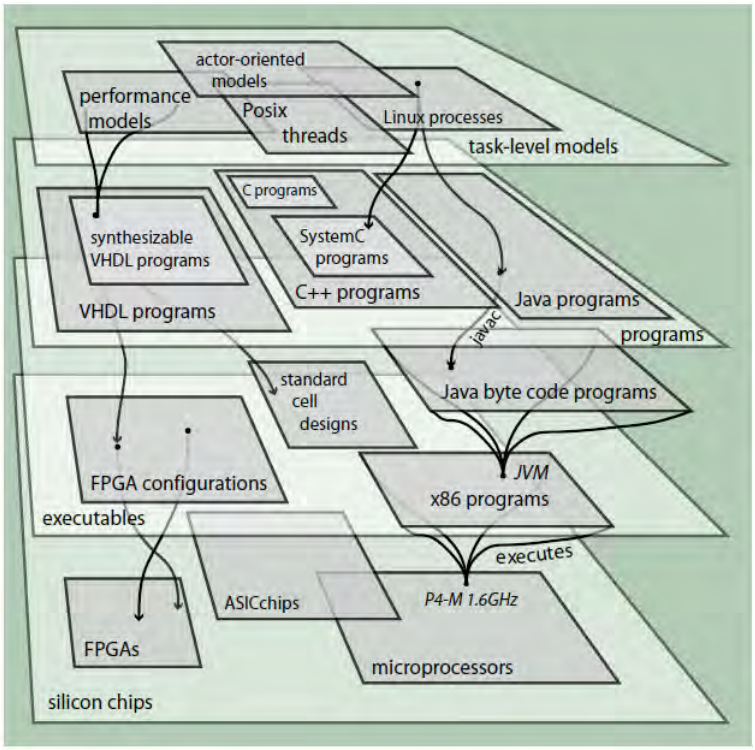

Figure 22. Computing Abstraction Layers. Source: Lee (2008).

An actor as a model of computation (Hewitt 1977) was first postulated by Carl Hewitt and formalized by Gul Agha (Agha 1986). In general, an actor (Hewitt 1977) communicates asynchronously and concurrently with other actors only through messages and has internal behavior and local state (i.e., does not share a global state). This makes the actor and inherent concurrent model of computation (MoC), as opposed to a Turing machine which is formulated as a single device acting on a sequence of discrete inputs. The Turing machine can be considered a specialization (Hewitt 1977) of the actor model. The actor model has been adapted for application to CPS and distinguished from other languages and methodologies (Lee and Neuendorffer 2004), particularly OOAD, as follows:

(1) Actors communicate (Lee 2003) with other actors in a model via ports and channels and have fixed parameters that configure its operation. A model of an actor is a hierarchical abstraction of the actor as shown in Figure 23. The model is an actor itself and is composed of actors.

(2) The model hierarchy (Lee 2003) has subclasses that inherit actors, ports and parameters of classes.

(3) Ptolemy II (Ptolemy II 2014) is a modeling and simulation tool based on the actor model. A "Director" in determines the MoC to include the communication mechanism and determines when to execute.

(4) Simulink and LabVIEW can be considered domain specific actor modeling languages. Simulink has a continuous-time semantic MoC and LabVIEW a dataflow semantic MoC.

(5) Software objects in OOAD invoke a method (Lee 2003) in a call/return sequence which requires a transfer of control. As Lee explains, this leads to "frail" composition where new components can break interactions and issues of managing threads of control such as deadlocks.

(6) SysML blocks linked via ports in internal block diagrams are "closely related" to actors (Ptolemy II 2014) but reflects a notational standard and do not unambiguously define behavior semantics.

(7) Actor models have abstract behavior types and exhibit behavior polymorphism (Lee 2003) as opposed OOAD object inheritance and abstract data types. The behavior is determined by the MoCs. The various MoCs supported by Ptolemy II are shown in Figure 24. Note that the actor model could implement a MoC of sequential untimed threads, but only as a deliberated design decision.

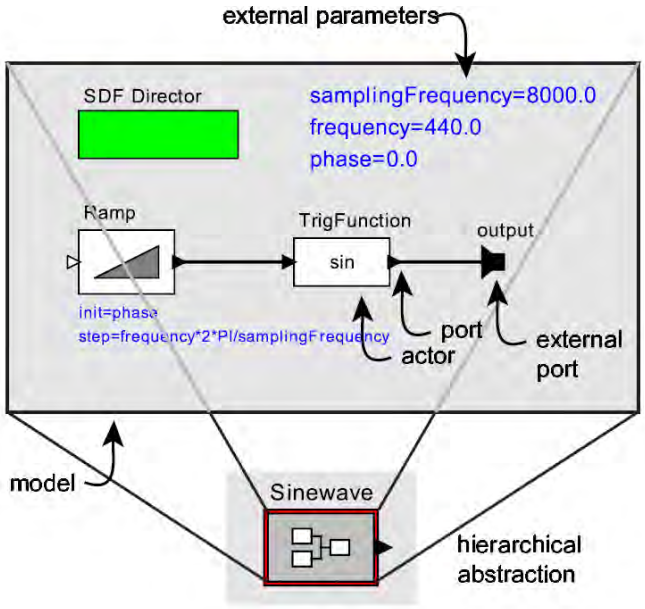

Figure 23. Actor Model and Abstraction Hierarchy. Source: Lee (2003).

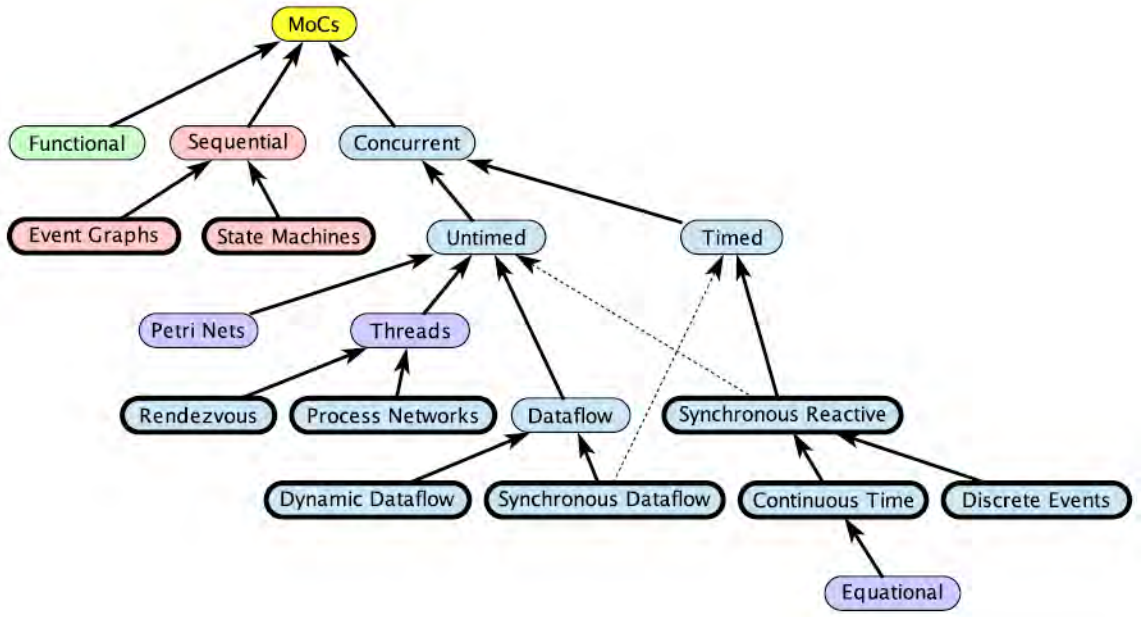

Figure 24. Ptolemy II Models of Computation. Source: Ptolemy II (2014).

The actor model shows much potential to support cyber-physical systems design, particularly if the "computational" model can be generalized to all types of execution so as to fully integrate physical non-computational components. However, there are not many examples of this, particularly for large scale systems. For large scale systems, it is not clear that reasoning about behaviors as actors is better or more intuitive than reasoning objects as things. These systems could have a hierarchy of behaviors with a hierarchy of networks and control. Higher level behaviors are managed as workflows via a graphical user interface (GUI) (Altintas et al. 2004) in Kepler as a companion to Ptolemy II. The reuse of both workflow and low level "task behaviors" that go on top the "architecture stack" is also unproven. It also seems to be more suitable for embodiment and detailed design than concept design which needs a more unifying principle and language across multiple engineering disciplines. Finally, it is not clear how agents or "context aware actors" would be accommodated and whether a unique MoC implementation or base behavior class would be required.

There are other efforts utilizing actor models. These include integration of a network simulator with Modelica (Al-Hammouri 2012), and object-oriented petri nets

(Ma, Fu, and Yu 2012). Modelica as an equation constraint language and petri nets constitute a singular MoC. The concerns of the Ptolemy II would apply similarly to these efforts or MoCs. However, they might find a niche as a more detailed design tool or as an analytical tool. It should also be noted that 4D/RCS by these definitions would be considered an actor model.

A meta-architecture approach (Rhajans et al.) with support from a general purpose architecture description language (ADL) called Acme (Garlan, Monroe, and Wile 2000) was defined to assess alternative architectures. An "architecture style" was created in AcmeStudio (Schmerl and Garlan 2004) utilizing "open and interconnected systems" concepts (Willems 2007) to include physical elements. Very broadly, Acme was designed to be a component based software ADL with multiple forms of connectors/interactions. One of its objections to OOAD is that objects have only one type of interconnection—"method invocation." It includes "architecture style" as one of its key ontological concepts to define a "vocabulary" that links system "families" together. For instance, a client-server architecture would be considered an architecture style. "Open and interconnected systems" concepts seemingly extend the Acme software-based components and connectors to include physical elements.

An instance of the architectural style is considered a base architecture (Rajhans et al. 2014). A base architecture for a collision avoidance system is shown in Figure 25. It is composed of multiple component types with multiple connector types appropriate to the connector per the specified architectural style. Multiple views can be specified relative to this base architecture to address various concerns. Architecture views shown (Rajhans et al. 2014) included verification for consistency assessment and analysis for "quality attribute" and trades assessment. This multi-view approach is considered an advantage (Rajhans et al. 2014) over multi-model approaches such as Ptolemy II. Disadvantages could be considered to be ecosystem support for a unique modeling language or "unifying framework" and CPS systems that might also include more "architectural styles" yet considered, such as large discrete event software and agents found in MIGVS.

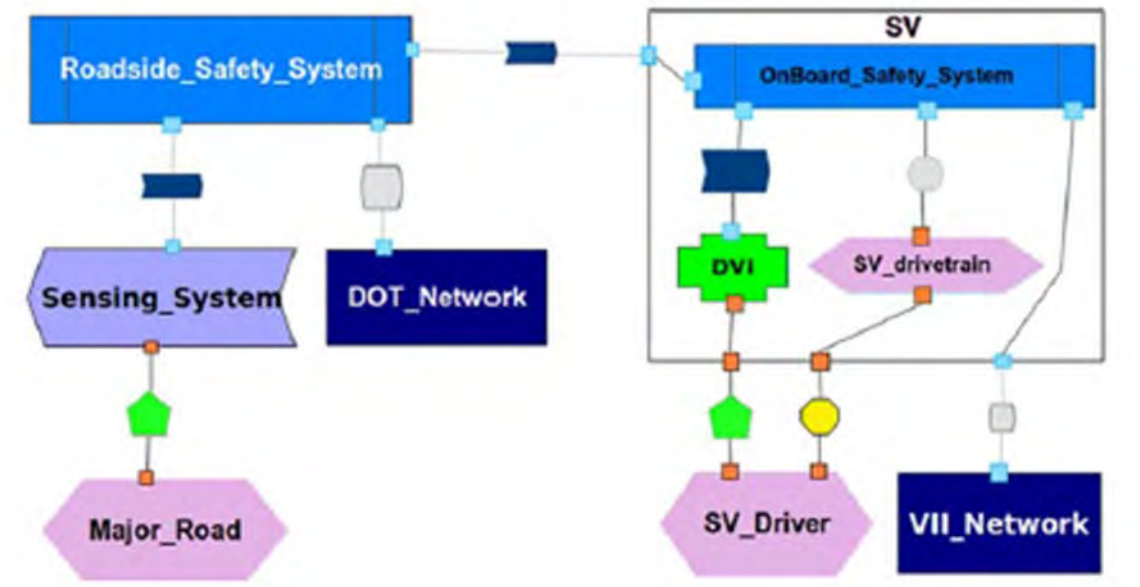

Figure 25. Base Architecture for Collision Avoidance System.
Source: Rajhans et al. (2014).

Unique CPS modeling languages need a way to define components and relationships. A semantic framework for MBSE (Delgoshaei, Austin, and Pertzborn 2014) uses a unique scripting language to link software and physical components, networks to model component relationships and can provide linkage with both to requirements via a semantic structure at multiple levels of hierarchy. This enables an integrated assessment of design concepts to satisfy requirements. It is not clear how well this support human interpretation from multiple disciplines is supported, how well intelligent components are addressed, or how well it addresses the interdependence of requirements

CyPhyML has a concept of a "design space" (Neema, Scott, and Bapty 2015) with multiple component alternatives. It also allows multiple domain models to define components and other aspects of a more detailed meta-model. The DARPA Advanced Vehicle Make (AVM) Program has defined a ground vehicle ontology and has looked at how the basic concepts could be extended (Lynch et al. 2016) to other CPS domains. This effort specifically aids the concept formulation and early systems engineering trade assessment. It does not appear to address higher level behaviors and associated

components nor does it enable object-oriented definition of discrete event software. These could presumably be added to the ontology, but it is not clear what in the "toolchain" is also required and how these unique environments are to be obtained and supported. Certain ontological structures could be readily established, but defining many types of relationships for a given domain prior to a development, may prove to be more difficult and time-consuming.

Extensions and profiles to more general purpose modeling languages tend to apply to the IoT type of CPS domain and their applications. A set of extensions to address CPS "process" have been proposed to the Business Process Modeling Notation (BPMN) 2.0 standard, referred to as "BPMN4CPS" (Graja et al. 2016). A CPS domain specific modeling language (DSML) (Aziz, Wagar, and Rashid 2016) has been proposed in terms of a meta-model implemented as a UML profile that enable its use with service-oriented computing. Another effort proposes a UML profile using a goal-oriented approach (Magureanu et al. 2010) "to handle the complexity of distributed applications and applies it to a gas distribution case study. A "cognitive architecture for IoT" (Sasidharan et al. 2014), to "conceal technological heterogeneity and provide services." This framework has a virtual object layer to enable lookup and registration of objects to support service discovery. A shared ontology and a Semantic Big Data Historian (SBDH) (Jirkovsky, Obitko, and Marik 2017) has been proposed to address and mitigate issue associated with semantic data heterogeneity, considered critical for Industry 4.0 success.

b. Quality Attributes

To understand attributes in the context of the overall CPS literature, requires some definitions. Attributes can be defined (Albus and Meyestel 2001) as "properties of an entity" that are typically measurable or as a characteristic of a person or thing. Per Albus and Meyestel, a characteristic can also be a behavior. Attributes that are measurable have values that may or may not vary. Albus and Meyestel further define a state as a condition or set of dynamic properties and a goal as a desired state of the world. It should be noted that goal-based agents will alter their behavior based on their belief and their estimate for achieving their goal state.

Properties can be defined (Garlan, Monroe, and Wile 2000) as "semantic information about a system and its components that go beyond structure" and are "extra-functional." Presumably then properties include all semantic information except for behavior logic. Constraints are "claims" on architectural design that "include allowable values on properties, topology, and design vocabulary." Structure is defined as the components and connectors and their topology or assembly. Structure consists of components and connectors, where components are "computational elements" and "data stores."

In an actor-oriented language (Ptolemy II 2014), the "semantics is largely orthogonal to the syntax, and is determined by a model of computation." Per Merriam-Webster: *Syntax* is a connected or orderly system: harmonious arrangement of parts or elements and *Semantics* is the study of meanings. The actor model semantics include the behavior logic and other properties that are provided via "parameters."

Given the above similar but different views of semantics and properties, the term *quality attributes* will be defined to include all system or entity semantics except behavior logic and physical properties. Types of quality attributes include (Barbacci et al. 1995):

(1) Performance

(2) Latency

(3) Throughput

(4) Capacity

(5) Modes

(6) Dependability

(7) Availability

(8) Reliability

(9) Maintainability

(10) Confidentiality

(11) Integrity

(12) Security

(13) Safety

More quality attributes can be added to this such as the aforementioned predictability, execution time, resiliency, modularity and (O'Brien, Bass, and Merson 2005): interoperability, usability, scalability, extensibility, adaptability, and modifiability. Even more attributes can be defined, combinations of other attributes can result in new attribute terms, and some current terms may be encompassed in combinations of other current terms.

Much of the literature (Khaitan and McCalley 2015) relative to quality attributes focuses on the IoT type of systems and for embodiment design, and does not identify any new quality attributes with the possible exception of cyber-security, though even that has been a concern of "CPS-like" systems. They potentially are a guide to points of emphasis and priority of quality attributes for CPS. A few examples include:

(1) Predictable and Reliable Performance—in particular the need to meet deterministic timelines (Derler, Lee, and Sangiovanni-Vincentelli 2011) or real time deadlines

(2) Cyber security—development of a context-dependent "role" security and trust model for data sharing (Stumpf, Bures, and Matena 2015)

(3) Safety—an approach to switching logic to safely manage modes in multi-modal systems (Jha et al. 2010)

(4) Modes—adapting the use of modes (Bures et al., 2016) as a "property" of a component in a smart cyber-physical system (Bures et al., 2015) to determine the best behavior in response to environmental uncertainty.

There are numerous other examples for these and other quality attributes. However, though they need to be addressed, quality attributes are not a good organizing principle for concept design. They are too numerous and somewhat nebulous when separated from a particular system or application, and not always measurable. For a domain like MIGVS, a certain priority can be achieved. For concept design, the priority needs to be those that define the trade space and need to be integrated with critical

structure and behavior. Both the system attributes and behavior cannot always be realized through allocation of attributes components, but often need to be augmented with additional components to meet the system attributes. Examples of this include nuclear event detectors and cable lockouts, redundant components, and anti-virus components.

c. System Applications

CPS system applications include (NIST 2013) smart: manufacturing, grids and utilities, buildings, transportation and mobility, and healthcare. Like with the design techniques, most system applications (Khaitan and McCalley 2015) have emphasized the IoT concerns. There are a few techniques for modeling local and cooperative behaviors (Loos, Platzer, and Nistor 2011) and complex behaviors (Ahmadi et al. 2011) that could apply to a MIGVS, but more as a behavior analytic technique than to model the system architecture. Direct CPS ground vehicle system applications include electric vehicles, autonomy and active safety, but are focused on an aspect of the system architecture and its iteration with design than on the overall architecture conceptualization. This includes an electrical and electronic (E/E) architecture and powertrain topology multi-layered design optimization scheme (Lukasiewycz et al. 2012) and a "functional-level co-design methodology" (Wan et al. 2017) to decouple architecture implementation and that includes design space exploration. The latter, in theory can scale up to a complete system architecture trade space, but reliance on functional and structured methods are likely to encounter the same difficulties that software analysis and design encountered as the behaviors become more complex, procedurally oriented, and data intensive. A base architecture (Rhajans) for cruise control has already been discussed

A "hierarchical information architecture" (Jobst and Prehofer 2016) to address vehicle CPS challenges for vehicles is shown if Figure 26. The intent is to organize information flow within, between, and across layers. The layers are spatial, hierarchy of control or behavior, and levels of information abstraction. It illustrates that within the vehicle and its local behavior, is a complex CPS with a behavior hierarchy including time and space relations, similar to 4D/RCS described previously. It also highlights the issue of data heterogeneity and interdependence. However, it is not clear how the layers are

meant to be integrated with each other and into the overall system architecture and design: is it a view, a software architecture, or a design pattern? Further illustrating the data heterogeneity issue relative to complex behavior, is a human interface and data fusion architecture (Wagh et al. 2011) that fuses on-board sensed and network obtained data to improve the driver's perception and reaction. It may work quite well in implementation, but it is a particular design architecture solution to a wider system architecture problem and trade space.

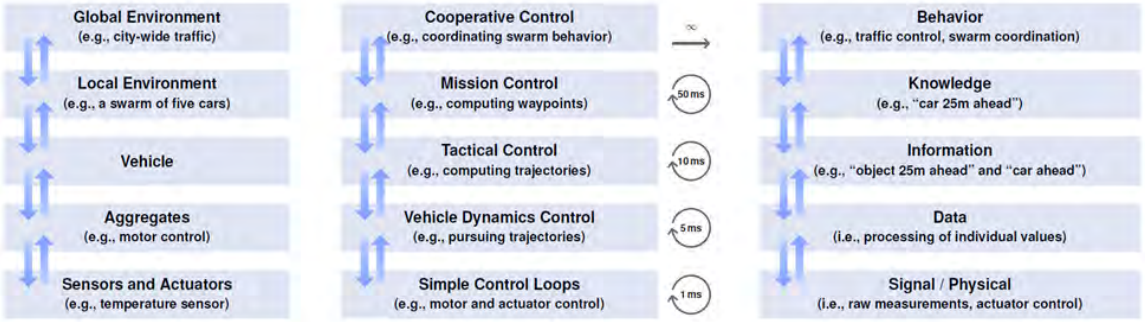

Figure 26. Hierarchical Information Architecture. Source: Jobst and Prehofer (2016).

C. SUMMARY

Most previous architecture and system modeling related research that can apply to military ground vehicle systems have not begun to incorporate the research and concepts emergent in the CPS domain. The interdependency of the cyber element and physical element within military ground vehicle systems has been increasing as had been the cyber dimension overall. Specific and formal techniques to address the interdependency, particular in concept design and trade space assessment, have not emerged or kept pace with advances in physical modeling. Most CPS and CPS-like architecture practice and research is focused on support of computational design, a distributed IoT type of system, or a particular slice of an application domain logic. Each of these research focus areas lack at least one or more dimensions critical to architecting systems in the MCPS and MIGVS domains. The critical dimension areas are: concept design, consideration of both cyber and physical components within an integrated trade space, intelligent aspects of control and behavior, sufficient use of hierarchy to manage complex behavior.

A focus on architecture to support CPS computational design is certainly needed. However, research in this area does not provide much indication of how it functions within a larger system or enterprise, where requirements for the CPS computation design originate, and how trade space analysis was conducted for both cyber and physical components so as to understand capability relative to cost and other constraints. This type of analysis needs to take place in a multi-disciplinary environment and then provide the appropriate concept baseline for embodiment and detailed designs. Multiple other types of design, such as 3D CAD design, need to be supported from a system concept baseline in addition to the computational design. The computational design research needs to be understood however to understand how best to inform it from the system concept baseline.

Architecture and modeling research for IoT types of systems mostly assume a very distributed network with a plethora of directly controlled devices by Internet applications. This in turn assumes a relatively flat control hierarchy. This is contrary to some research on smart and/or autonomous systems that utilize several layers of control above the direct control of sensors and actuators. The hierarchy of control not only reflects a hierarchy of behavior, but of different types of behavior, or at least different emphasis of behavior types at the various levels. Some IoT research has addressed some high level or relatively intelligent behavior, but it is reflected as workflows or services to be managed by humans or in a single application layer. The IoT research also does not address all the constraints introduced by a MCPS, though some effort has been focused on addressing a dynamic environment or context awareness. These approaches are also likely to be limited relative to the most dynamic aspects of the environment if managed by a distributed Internet-like network with distributed applications acting as a single layer of control.

Research in the application domains of large non-mobile infrastructures and/or controlled external environment do not address the full range and interdependency of multiple forms of logic, performance, and constraints identified for the MIGVS domain. CPS research for ground vehicles systems that rely on supporting transportation infrastructure or niche areas, such as hybrid electrical control design, do not address

military unique aspects of the MIGVS domain. These include the operator or intelligent interactions and control, the wider range and complexity of the dynamic environment, and the range of constraints required for integrated concept design and an integrated trade space.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AGENT AND OBJECT ORIENTED MODEL-BASED CONCEPT DESIGN FOR MOBILE CYBER-PHYSICAL SYSTEMS

The system behavior logic is modeled as a set of abstract classes and objects that have an internal behavior and that interact with each other as well as the external environment. The higher level system behavior is determined by the object interactions. The behavior logic transforms the entire set of system inputs to the entire set of system outputs subject to the system and component properties and constrains. A system with intelligent behavior is a context aware or smart CPS. It can perceive external events through its input, decide which behavior is required, and assess whether its output has produced the necessary effect. It can then decide what further action or behavior is required and execute that behavior consistent with its design and constraints. As such, special emphasis to how the context is modeled and understood by specialized objects, called agents, is critical to the system architecture and trade space.

The following terms and definitions are considered fundamental and critical to describing a system architecture and will be utilized throughout:

(1) *System Syntax*—a set of components and connectors, their arrangements, structural relationships, and their abstractions. A system syntax can have multiple types of abstraction for multiple types of arrangement or views. For example, a system can have logical abstractions focused on the interaction and behavior of components, and physical abstractions focused on how the same components are physically assembled.

(2) *Components*—are the physical elements of the system to include software and hardware. When physical components are realized as a particular physical instance of a generalized abstraction, they will be referred to as objects.

(3) *Connectors*—represent the interactions between components (Garlan, Monroe, and Wile 2000) and their abstractions. Connectors may represent physical instantiations or logical abstractions not distinctly identifiable from the components they are associated with. Connector abstractions will be defined as *ports*.

(4) *Interactions*—the exchange of energy, material or information between components and the external environment and/or between components

themselves. The components associated abstractions can have abstract interactions. Interactions are conveyed through connectors or ports.

(5) *System Semantics*—is the meaning or purpose of a system and its components/connectors beyond its syntax or structure. It consists of behavior logic and properties.

(6) *Behavior Logic*—a set of interactions of a system or component over time. Execution of the logic at a point in time is dependent on the specific incoming interaction and the value of the properties at that point in time. Behavior relates to a system and component's purpose or functionality.

(7) *Properties*—consist of physical attributes and quality attributes that typically have values or a range of values.

(8) *Physical Attributes*—symbols and their values that pertain to the geometry, mass or other physical aspects of a system, component or physical connector.

(9) *Quality Attributes (QAs)*—non-physical attributes that further specify a system, component or connector, their behavior logic and the behavior logic's execution. QAs of focus in this research include performance, reliability, information/data and world/goal states.

(10) *Constraints* (Garlan, Monroe, and Wile 2000)—"claims" about a system or component's syntax and semantics "that should remain true over time." "Typical constraints include restrictions on allowable values of properties, topology and design vocabulary."

The concepts to be discussed are generalized and defined independent of a specific modeling language as much as practicable. Key concepts are expressed as "reference models" that can be instantiated and elaborated for a given system or project. Given the emphasis on modeling and systems, SysML will be utilized where required for concepts that have more intricate semantics and syntax, such as objects that have generalization and aggregation relationships. This is due more to its ready availability than its advantages or disadvantages as a modeling language.

A. MOBILE CYBER-PHYSICAL SYSTEM LOGICAL STRUCTURE AND BEHAVIOR CONCEPTS

Figure 27 shows the Mobile Cyber-Physical System (MCPS) Architecture Concept Data Meta Model (DM2). This model captures the critical concepts and relations

required to model the behavior of a MCPS, and is defined so as to enable a direct compare and contrast of the DODAF concept DM2 of Figure 4 and its critical concepts.

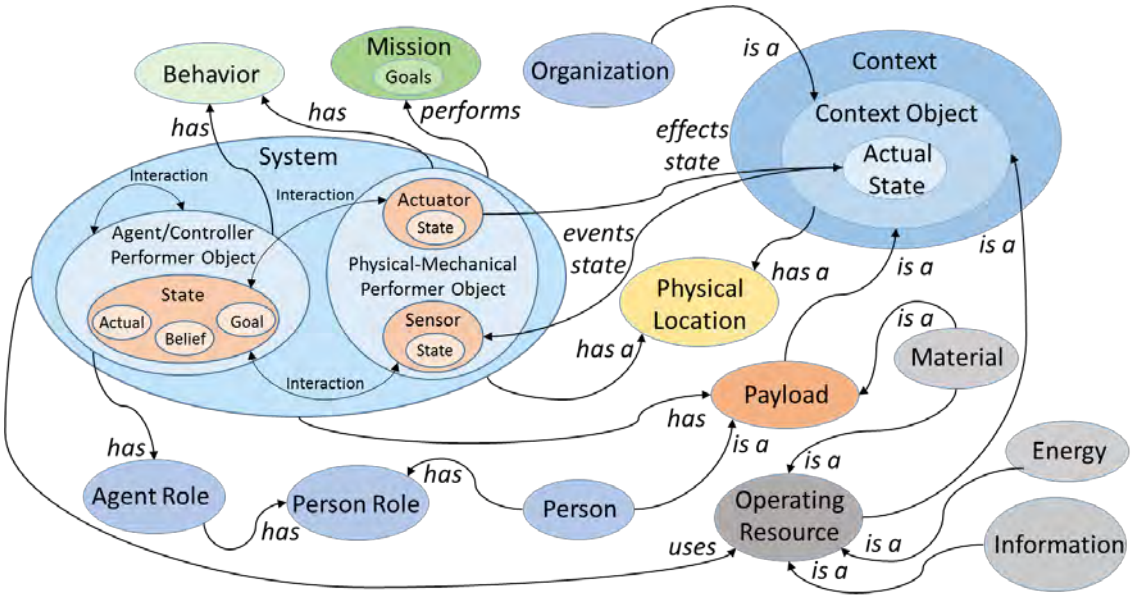

Figure 27. MCPS Architecture Data Meta Model

Figure 27 shows that the world is composed of instances of real-world things or entities called objects. These objects have state and can be grouped into a system or into the context (i.e., objects that are relevant to but not "part" of a system). System objects are performers that have behavior and interact with each other and context objects to detect and effect state needed to perform missions and achieve goals. System objects that directly detect or effect context state are typed as sensor and actuator objects. They are controlled by other system performer objects which include agent performer objects, a type of object whose behavior is driven by its current or believed state of the world relative to its goals or goal states. Context objects can have intrinsic time-varying properties and/or static properties that time-vary relative to an MCPS due to its changing physical proximity. Context objects also include Payload and Operating Resource, two types of objects that are not "part" of the system, but are at times physically located within the system boundary.

Comparing the DODAF DM2 model of Figure 4, there is virtually no consideration of the context versus an explicit model of the context as means of modeling external events and effects that the system must react and act upon as well as track the state of context objects. The context is a required as part of the architecture concept data model as well as required data used by the system. An organization is potentially just one type of object in the context that interacts with the system. Activities and services are types of behavior that may or may not be utilized by the system. Rules and conditions can be incorporated as mission goals or a kind of limit or tolerance on mission goals. Physical location is always important, but geolocation or location referenced to the world may or may not be. Information and material are augmented by energy and represent externally supplied operating resources to the extent they are of interest to the system and its architecture. Material is also a type of Payload. Finally, capabilities are the missions and goals that the system performer objects are capable of performing and reflect the desired state of a resource, but most distinctly the desired state of the context or its state relative to the system.

1. System Performer Object

Any component or part of a MCPS can be classified as a performer object. All technology can be thought to have an executable (Arthur 2009), even something as seemingly static as a bridge or a support beam. An executable is one way to view a behavior. Technology is built to a purpose or mission, and if it meets its design semantics or purpose, it has executed effectively. A computation or computing technology represents only one form of executable. A human operator can also be considered an executable. Conversely, things that seemingly represent only an executable, such as software, can also be thought of or modeled as a certain form of technology or physical entity. Each technology as applied to a system can be viewed as a component at various levels of abstraction that have behavior and interactions and particular instances that are system objects. System objects can be distinguished by the nature of its executable and the nature of its interaction.

Figure 28 shows a general model of a system performer object. A *system performer object* is an abstraction of a component that can interact with the external environment and/or other system objects via connectors or ports; where the interactions involve energy, material or information, where the interactions are directional, and where the object has identity, behavior logic and state. The object behavior at this level of abstraction is a model of its executable. As such, it can be an abstraction of a software object similar to objects in software OOAD except that it does not necessarily invoke a method. It can be an abstraction like an actor model communicating with messages except the behavior is not necessarily polymorphic, or it can be an abstraction of any physical component that interacts via energy or material. The term system object is utilized because of its close association with real-world things.

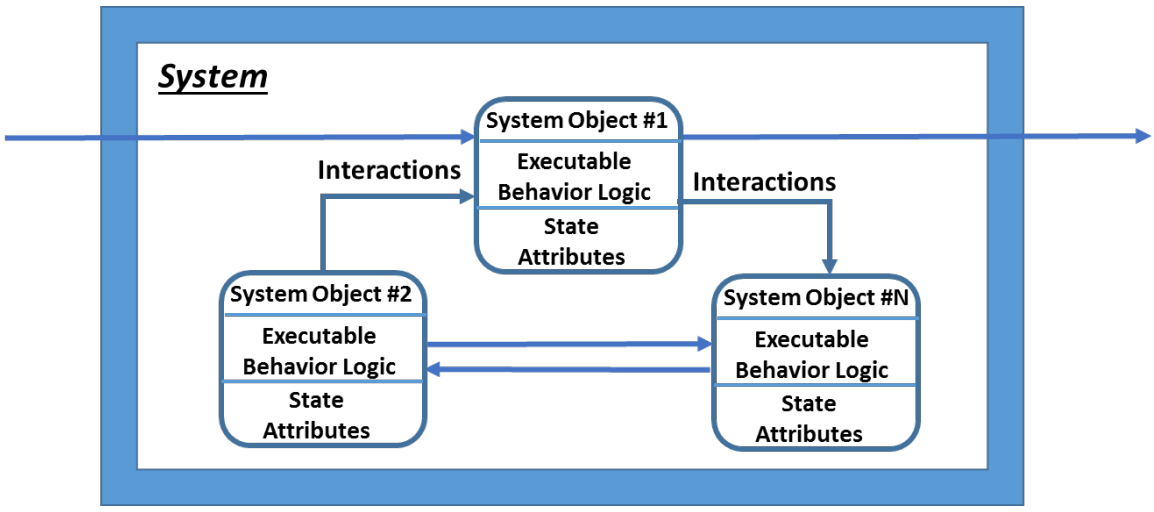

Figure 28. General System Performer Object Model

Performer objects can be grouped and arranged in a hierarchy to address complexity. This logical object hierarchy, like social hierarchies, are grouped by interaction rather than spatial proximity (Simon 1962). System Performer Objects are at the top level of this hierarchy and can be recursively decomposed into sub-performer objects or even sub-sub-performer objects. As Simon explains, objects within the same hierarchical group interact more than objects in different groups. Objects that do not or minimally interact with each other would have different "base" system performer objects.

The MCPS domain can be divided into two types of sub-performer objects: active and resource. These two types of sub-performer objects will aid in creating patterns across performer object classes.

Active performer objects can alter and intelligently assess their impact to the context. The impact and sensing of the context is subject to computational control for the MCPS domain. The MCPS domain has at least a two level hierarchy of computational control: cyber-physical and intelligent. Intelligent control computation can itself has multiple levels of logical or cyber hierarchy depending on the complexity of the system. Active performer objects are defined as follows:

(1) External Sensors/Actuators—objects/components that exchange energy, material, or signals between the system and the external environment. Sensors convert "stimuli" (Poole and Mackworth 2010) from the external environment into information and actuators that convert information into actions to the external environment, these actions can include signal transmission. Similar objects that do not interact with the external environment are categorized as part of various system resources.

(2) Direct Sensor/Actuator Control—objects/components that provide direct control or first level hierarchical control of external and internal sensor and actuator objects/components. This control is based on position feedback of the sensor/actuator and is not based on any context awareness. Component types include servos, regulators, feedback control, etc. Note that external sensors may have direct control that includes feedback in addition to its primary purpose of providing information about the external environment to higher level performer objects.

(3) Agent Logical Object—an agent as an object/component requires extensive discussion and is addressed subsequently.

Resource objects/components address certain constraints introduced by the mobility of an MCPS and/or limitations of engineered systems. Resources must address these constraints and provide support to other components that require the resource. The types above are consumers of one or more of the resource sub-types. Interactions, connectors and ports reflect the types of resource exchanged. Resource objects are defined as follows:

(1) Physical Structural Support and Protection—objects/components that provide primary and secondary structural support to all system objects/components as well as protection against the direct effects of the external environment. Component types include mounts, brackets, glacis, walls, and electronic chassis' or bays.

(2) Power and Energy—objects/components that generate, transform, store, and/or distribute power or energy to components that consume it and that provide the prime automotive force of the mobile system.

(3) Material Support—objects/components that generate, transform, store or distribute material required by other objects/components. Relevant material in an MIGVS can include fuel, oil, and ammunition.

(4) Information/Computation—objects/components that generate, transform, store or distribute information required by object/components. Component types include computers, computational stacks, application software support, networks, and resource access firmware.

From a behavior logic standpoint, both active and resource object types encapsulate a behavior executable that interact logically through ports with each other and/or the external environment to comprise the entire system behavior logic. Again, even structural objects have a behavior logic, though perhaps not as dynamic as others. The logic of a structural support will be determined by its reliability (e.g., working, not working, degraded), and could be relatively dynamic if one considers time-dependent reliability. All performer objects have measurable behavior based on the time-varying change in values of its state attributes. These state attribute values change based on the interactions of performer objects with each other and the objects in the context. Interaction types are categorized as follows:

(1) Energy—interactions that provide or dissipate power, generate or absorb a force, support an equilibrium or provide a signal.

(2) Material—interactions that support the exchange of a solid, liquid or gas.

(3) Information—interactions that support the exchange of data, information, or knowledge.

Energy and material interactions between two objects are direct and of the same type, though the energy/material may be transformed to a different type by the object for

a direct interaction with another object. Information interactions however can be viewed as having two types:

(4) *Direct logical interactions* this is an abstraction of a direct and same type of interactions. Certain objects can transform the information from one type to another to facilitate some end objective, such as signal to data transformation by a device driver.

(5) *Indirect logical interactions* are like representations of information indirectly exchanged between a source and a consumer, such as between two software applications.

Objects that exchange information need to support both these types of exchanges.

2. Context Object

As indicated in Figure 2, a MCPS interacts directly with its external environment through signal, material and/or energy and interacts indirectly through information exchange. Consideration of the system, its boundary, and its direct interaction with the external environment or its context, is an important and typical consideration within systems engineering. However, as indicated in Figure 27, a MCPS with a cyber hierarchy that has indirect logical interactions with the Context, understanding and modeling that Context takes on much greater importance. How well or whether the correct behavior gets executed by a system is dependent on the accuracy of the information relative to the context.

The traditional view in systems engineering is that the system's context or external environment drives system inputs and is subject to system outputs. The systems' behavior logic translates the inputs into outputs. This is still the case for a MCPS, except that there is a wider thread of behavior logic the intelligence of the system has to address, namely event to effect. *Events* are a change in world state that can cause the system to execute some appropriate behavior when perceived by the system. *Effects* are a change in world state actuated by the system consistent with some goal or purpose. A MCPS must detect events and assess the success of failure of "desired" effects in the context or external world. A standard control system by itself is not aware of events and effects in the context, but rather reacts to an input from with the system, couples it with knowledge

of the plant that is controlling, and produces a new output to control the plant. The control system is not aware of the effect that the new output has produced. An intelligent system, responds to an event in the context and produces an effect in the context and then determines whether that effect was satisfactory.

Defining the world in objects and interactions enable focused reasoning about a limited portion of the world state relevant to given problem. It is not practical (Poole and Mackworth 2010) to reason about the world in terms of all possible states, there are too many. However, the state of the world can be abstracted, particularly for concept design, to a set of things or objects, each with a set of attributes, that when defined can be considered the relevant state of the world. The first abstraction is to divide the world state into the system and its external context. As previously discussed, the system can be defined as a set of performer objects, each with a set of attributes. The system's state is the set of objects with all its attributes defined at a given moment in time. Correspondingly, the Context can be defined as a set of objects with attributes that change state, at least relative to the system. As a domain, there may be set of top level classes that an MCPS might always need to address (e.g., terrain). These classes can be further decomposed into lower level classes or objects along with lower level but related attributes as needed (e.g., roads with lane attribution like marking type, color etc.).

3. System Connected Object

A System Connected Object is context object that can be stored, housed and/or used within or at the system boundary. There are four types of MCPS System Connected Objects:

a. Operating Resource

An operating resource is an object that used by the system in its operation. It is not a "part" of the system, but is necessary to its operation and the capacity to carry the resource is part of the system. There is generally a way to bring an operating resource into the vehicle and store until it is needed. It is generated in the context and then consumed in use.

(1) Physical. Physical abstraction is introduced here so as not to bias the solution for MCPS concept design. For example, a gasoline powered engine would use material whereas an electric vehicle would use energy. This can be abstracted to a Physical object with an attribute of operating range until the specific technological approach is selected.

(2) Material. A material store object, like a material interaction refers to a gas, liquid or solid. Examples include fuel, oil, and ammunition.

(3) Energy. An energy store object is purely an abstraction object and will not be elaborated for concept design. Once a technological solution is established, the energy object is likely to show up as an attribute of another object (e.g., battery charge).

(4) Information. A Context information object is externally generated information provided to the system for its use. Examples include maps, precision geolocation, and information about the Context beyond the system's sensors.

b. Payload

A Payload Object is defined here as an object that a MCPS carries during or as part of its operation and that is not part of the system or in any way consumed by the system.

(1) Person. A person object is a human on-board the system that must be physically accommodated or carried aboard the system.

(2) Operator. Operator(s) must be physical accommodated so as to be capable of operating the system.

(3) Passenger. A person object not directly involved in the operation of the system (e.g., infantry squad).

(4) Cargo. Cargo is an object that the system is specifically designed to carry and can be endemic to the system mission (e.g., a dump truck). It also includes personal equipment need by any operators or passengers of the system.

(5) Operating Resource Augment. These objects are the same as Operating Resource Material Objects (e.g., fuel, oil), but are not part of the systems' design capacity. An example would be extra containers of fuel stowed somewhere aboard the system.

c. Person/Person Role

Person Object can both be defined as a payload and in terms of its role in the operation of the system. Any operator of the system should have attributes of both these classes. However, in the initial concept design of a MCPS, the operator as a person should not be presumed but rather included in the trade space for final concept design. The person role will be accommodated by agent roles and agent logical objects and addressed subsequently.

d. Tactical Network

The tactical network is an external object consisting of at least one physical channel. An MCPS by definition can be thought to connect to at least one external network via an embedded network interface. External networks can control, guide, or augment the operation of the system, support system test and diagnostics, and/or support system upgrades.

4. Mission/Tasks/Desired Trajectory/Goals

A goal is a desired future state of the world (Albus and Meystel 2001). Albus and Meystel distinguish between two types of goals relative to behavior: 1) maintain a system relative world state over time or 2) achieve a change in that world state. A *goal* is a physical world state or knowledge state to be achieve or maintained for a given time. Albus and Meystel further define a reference trajectory as a set of goals along a timeline. Here a reference trajectory will be referred to as a *desired trajectory* and defined as a desired path through a given state space beginning with an initial state and culminating in an end goal state. A *trajectory* is any path through a given state space. The *state space* is the set of possible attribute values of a set of one or more System Performer or Context objects. A desired trajectory as a minimum has an initial state and an end state, but can also reflect multiple interim goals and can be recursively decomposed into low level desired trajectories as shown in Figure 29. Goal 1 reflects the initial state and Goal 2 the end goal state for the sub-trajectory.

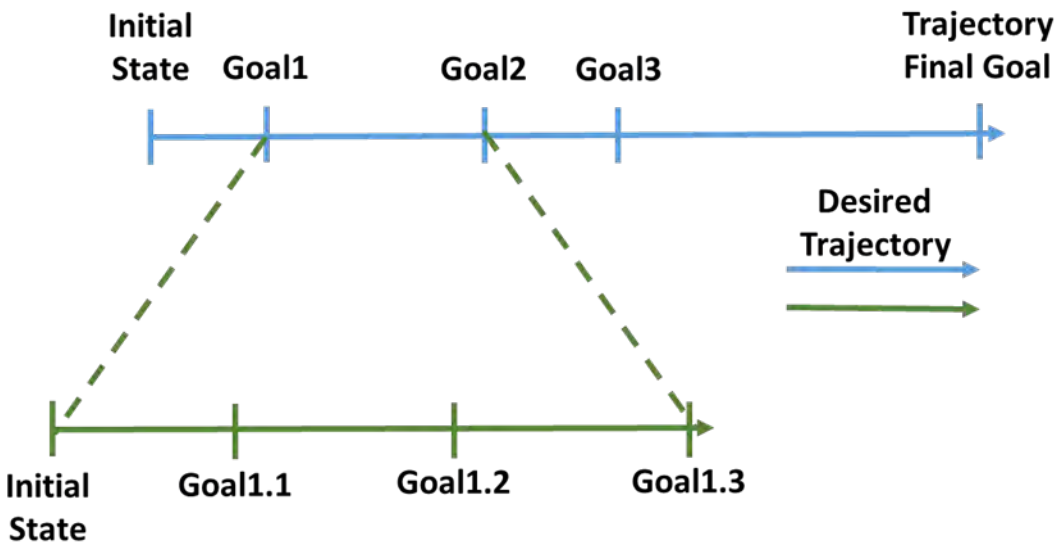

Figure 29. Hierarchy of Goals and Reference Trajectories

The goal definition for both Achieve and Maintain goals need to be definitively specified if the intelligent behavior is to be executed by machine. A *goal specification* is defined as a canonical set of goal measures for a well-specified goal. The a goal is specified expressed in terms of four measures and two value functions as shown in Table 2. Each type of goal has both a time and state target, a time and state tolerance relative to the target, and value functions that compute the loss of value as actual time and state measures vary from the assigned target. The value functions shown in Table 2 are based on a normal probability density function (PDF) for an achieve goal and an exponential cumulative distribution function (CDF) for a maintain goal. These distribution functions are notional and actual distributions could be defined experimentally and augmented with constants.

Table 2. Achieve and Maintain Goal Measures

Measure	Achieve Goal	Maintain Goal
$tTar$	Assigned time to achieve or complete	Assigned time to maintain
$tTol$	The acceptable variation around $tTar$.	The acceptable variation around $tTar$.
fv(t)	The value loss around tTar computed as $e(-(tAct - tTar)^2)$ where $tAct$ is actual current or completion time	The value loss around $tTar$ computed as $e(-\left(\frac{tOutSt}{tInSt}\right))$ where $tOutSt$ and $tInSt$ are the total time out of and in state, respectively
$sTar$	Assigned target state measure.	Assigned target state measure
$sTol$	The acceptable variation around $sTar$.	The acceptable variation around $sTar$.
fv(s)	The value loss around $sTar$ computed as $e(-(sAct - sTar)^2)$ where $sAct$ is actual current or completion state	The value loss around $sTar$ computed as $e(-\left(\frac{(sAct-sTar)}{sTar}\right))$ where $sAct$ is the average, cumulative or current state over a maintain time interval

Tolerances can be expressed as state and time variables or as a measure relative to a target of one. For complex state variables (e.g., a list with many items), it may be more convenient to express the target as a composite value of one with a separate state equation comparing the target list to an actual number of items in a distance function. Also, a maintain goal may be expressed without a time value since the desired state is to be maintained for an indefinite period of time or until the system is commanded to do pursue a different goal. However, the time in state versus out of state may impact its overall end goal state satisfaction. In this case, the goal is to always maintain with a value of one.

A desired trajectory or behavior with a goal is unlikely to exactly fail or exactly succeed. The behavior has a loss of value to the degree that goal state and time targets are not met. A behavior in response to a desired trajectory can be deemed a failure by a

superior agent based on the loss of value, such as an actual time or state beyond the specified tolerance. A decision at that time may be made to proceed with the mission, alter the mission, or abort. A deemed failure to meet a goal may require a special classification when compared with hardware failures and standard concepts of system reliability (e.g., "essential task soft failure") as compared to an essential function failure. From an overall requirements standpoint, goal requirements can be listed as narrative with other system functional requirements. The goal requirement narrative would identify standard state and time targets, standard state and time tolerances, and ideally include an agreed to loss of value function. However, actual failure would be determined dynamically based on mission conditions.

As indicated, a goal can express a desired physical state or a desired knowledge state. The goal of certain military missions and tasks is to acquire some state of knowledge. *Knowledge state* is defined as the current state variable values of events and situations. An *event* is defined as a temporal occurrence of interest or discovery of an object's attributes and a *situation* is defined as "a relationship that exists between entities and events in space and time" (Albus and Meystel 2001). For example, an earthquake is an event, but someone at the top floor of a twenty story building at the epicenter, is a situation. Events themselves can object attributes that are static, but are of temporal interest to a system, particularly a MCPS. A knowledge goal is the determination of the state; that is, the desired state of knowledge is to know the state rather some particular state. The same goal specification measures can be used, though the detailed expression calculation may be different.

A mission reflects the fundamental or overarching purpose of the system. A *mission* is defined as an ordered set of tasks. A *task* is defined as a set of trajectories. Most if not all MCPS are designed to achieve some effect in the environment as its overarching purpose and can be considered to be the *mission effect desired trajectory* as its object identity. However, most MCPS also have constraints on safe and reliable operation as well as constraints on its operating resource usage. These constraints can be defined as desired trajectories over the same mission trajectory timeline as notionally shown in Figure 30. The advantage of viewing these explicit desired trajectories relative

to the mission is that the goals constraints can be adjusted to reflect different rules and conditions. For instance, a critical mission of moving a passenger to a hospital emergency room can be weighted more heavily by allowing more leeway to the goal constraints of safe operation. The operating resource and operating desired trajectories can be generalized as normal or disciplined operation desired trajectory. Many more of these types of trajectories could be defined.

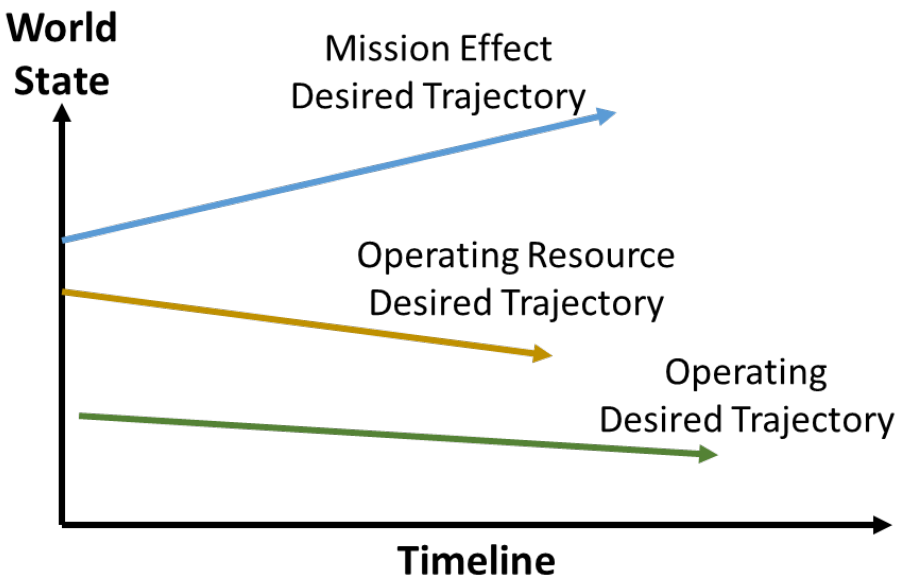

Figure 30. Notional Mission Trajectories Graph

The system and context or world objects/attributes of the mission and all its trajectories reflect the entire world state of interest to the system and can be executed concurrently. The trajectories may or may not share world objects of interest or a particular subset of the overall state variables. The actual direction of trajectories shown in Figure 30 are notional and merely reflect that the state variable change values in a particular way of time. Additionally, there is a third type or class of trajectory not shown that can be defined as *exception handling desired trajectory*. These can be explicitly defined as abnormal or "non-sunny or rainy day" operation and have goal constraints that can be adjusted as needed within design constraints to reflect different rules and conditions. For example, an intelligent ground vehicle system needs to detect dynamic

obstacles in the context and take proper action. Once a dynamic obstacle has been detected, the appropriate action may vary depending on its risk to the overarching mission effect.

Mission composition of mission effect, disciplined operation, and exception handling desired trajectories and their goal specification measures, define the measures of effectiveness (MOEs) and measures of performance (MOPs) for the system. The trajectories can be further typed and decomposed. The corresponding decomposition of world objects, state attributes and goals result in a hierarchy of MOE/MOPs. The system and context objects can be decomposed as needed along with the corresponding state space and be defined as a *world state model*. The world state model will reflect a set of attributed data objects for concept design. The system's assigned and derived missions will selectively reference the data objects from the world state model and constrain their attribute values as needed to meet goals. The goals are much like use cases except they do not presume an operator and are embedded in the concept design of the system.

5. Agent Logical Object

Agent logical objects (ALOs) are system's objects capable of intelligent behavior. An ALO is capable of perceiving its environment and achieving or maintain an effect in the environment (Russell and Norvig 2003). Like other system performer objects, ALOs have a set of interactions, a behavior logic, and a set of state attributes. Also like other performer objects, agents have identity and represent a solution independent abstraction of some physical element or technology. In this case, an ALO represents an abstraction of intelligent logic that can be realized by person's brain, application software, or a technology like a neural net. It is independent from, but requires integration or transformation access (e.g., some direct logical interaction path) to some type of compatible computational execution engine framework (e.g., a computer for the application software). ALOs can be aggregated along with computers, control systems, and sensors/actuators to form embodied physical agents, can be realized as distributed software, and/or can be physically realized by a person.

As shown in Figure 31, ALOs logically interact via commands and percepts. These are indirect logic interactions (e.g., application to application). They logically interact only with other ALOs or sensor/actuator controllers and execute within an application control hierarchy. They receive commands from "superior" agents or agents that are higher in the control hierarchy, and issue or generate commands to subordinate agents. Conversely, they issue or generate percepts to superior agents. Each ALO has a world state of interest composed of entities/attributes that reflects the world state model discussed previously. The current value of the attributes reflects the ALO's belief state. The goal state is composed of a set of constrained values on the world state space ordered into trajectories that reflect the assigned and derived missions discussed previously. The ALO behavior logic considers the belief state relative to the goals state and takes appropriate action.

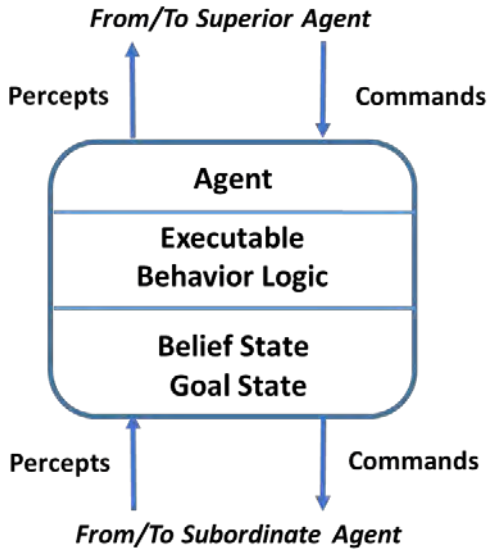

Figure 31. Agent Logical Object (ALO) Pattern

The ALO high level internal behavior is shown in Figure 32. The behavior block executes on the data in the goal and world states. The agent behavior processes command from the superior agent and generates commands to subordinate agents required to meet the command. It receives status back from subordinate agent percepts and provides its

collective status up to the superior agent. The superior agent command has an assigned mission with a set of trajectories which in turn has a set of goals. The goals constrain a set of world state entities to desired values. The ALO updates its world state based on subordinate agent percepts and compares it against its overall goal state. Based on the delta the agent can decide if any new behavior or subordinate commands need to be generated. Regardless, it reports its current state up to the superior agent. If the ALO is realized by machine, the world state model and the goal constraints would represent logical data stores and would be incorporated into a concept data model for the system.

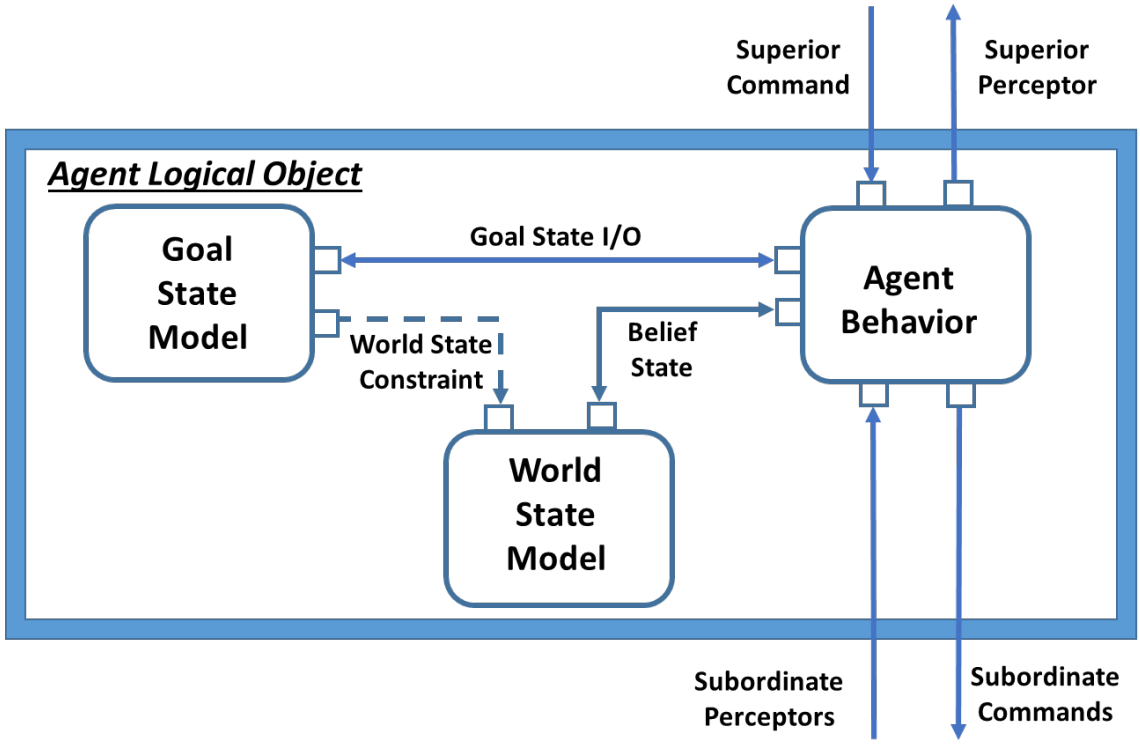

Figure 32. ALO Internal Behavior

The indirect logical control and interaction topological pattern of ALO's for a MCPS is shown in Figure 33. This pattern begins the formulation of each ALO's identity. The general behavior pattern above is repeated for each of the ALOs. The ALO types in this hierarchy are defined as follows:

(1) Mission Agent (MA)—controls the overall mission or operation of the system. It initiates, orchestrates and terminates the top level tasks or trajectories. If connected to a larger network or systems of systems it will also respond to commands to a superior agent and provide percepts to that agent.

(2) Task Agent—executes the top level task or trajectories. Responds to percepts with world state information from one or more detection agents, determines an appropriate course of action, and generates a command to a single intelligent control agent.

(3) Detection Agent—interprets world state from data received from sensors and sends percepts to its higher level task agent. Detection agents are the primary means for directly interpreting information about the state of the world. The number of detection agents will correspond to the different types of sensors needed to determine various unique aspects of the world state. The detection agents control the sensors as required to detect the world state.

(4) Intelligent Control Agent—responds to commands from the task agents and issues the necessary set of coordinated commands to the systems actuators. This is the primary means to bring about an effect in the external environment or change to world state. There is a single ICA to insure all control systems receive a coordinated set of inputs.

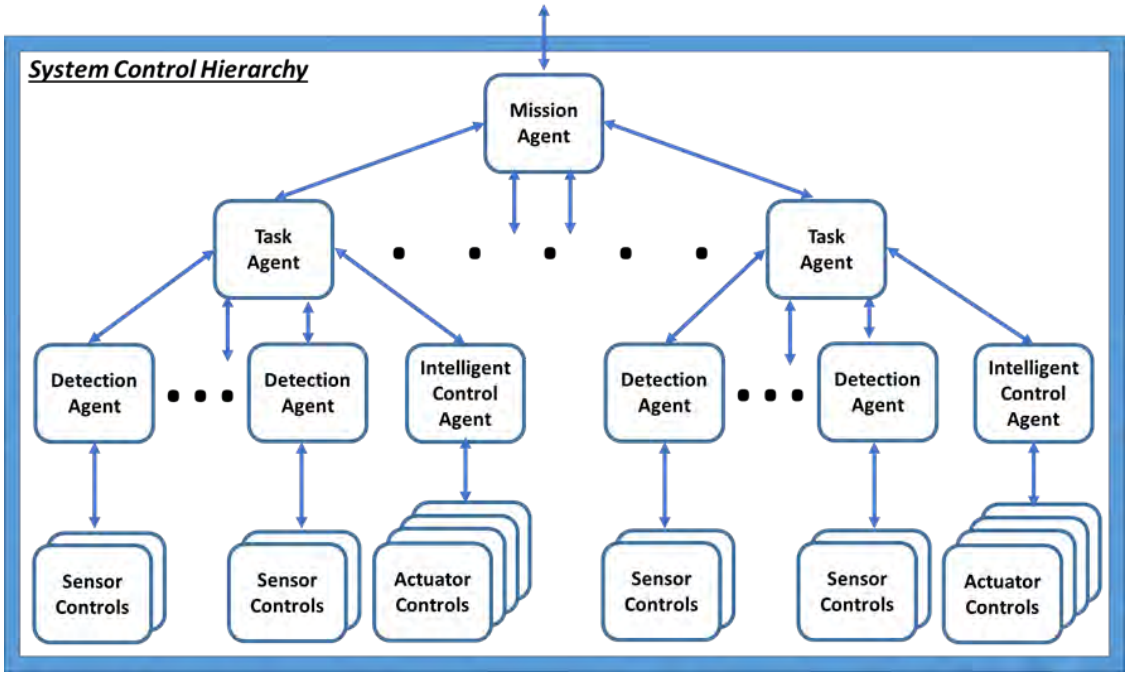

Figure 33. Agent Logical Control Hierarchy

For a U.S. Army MIGVS, the mission plan is a highly detailed and coordinated a priori plan through the entire mission timeline. It has an overarching purpose, but also many "disciplined operation" trajectory type tasks or rules as well as belief state information about the context. Discipline operations examples include following a route plan, maintain situational awareness, and proper communications. Each ALO in the hierarchy would have some portion of the mission plan assigned and would also have to decompose the plan into sub-trajectories and/or sub-goals. Assuming the plan does not change during the mission, the ALO percepts are reporting on the status against their portion of the mission plan and the mission ALO reports this back to the unit or task force commander. This does require identifying which part of the plan is currently active, hence both trajectories and goals can be considered to have a state.

The task agents, and the detection and intelligent control agents in particular, not only have perform as appropriate to the a prior plan, but also have to dynamically plan to meet dynamic events. There are two types of dynamic events that must be considered. *Unplanned events* are dynamic events that may interrupt any given desired trajectory, at each temporarily. Each ALO must be able to handle a finite set of exceptions to some performance standard or goal, and then be able to plan and execute a recovery to its original plan or dynamically adjust to a newly assigned desired trajectory or goal changes. *Unplannable events* are dynamic events that do not necessarily interrupt the a priori reference trajectory, but cannot practically be planned prior to execution. They are dynamically planned during execution and then executed within the desired trajectory timeline. Basic vehicle movements for instance cannot be practically planned prior to beginning a mission.

To complete the ALO identity, each ALO must be assigned a person role and a place within a top level system performer object. Humans perform multiple roles where each role has an overarching purpose. For each given purpose, a human will execute a set of behaviors that will at least have some unique aspects to it, though a given behavior can correspond to multiple roles. A *role* is defined as an overarching purpose that has an associated set of intelligent behaviors that will execute in response to external events or stimuli. A different role may execute behavior differently to the same event or stimuli. A

given agent logical object corresponds uniquely to a given role. In terms of component realizations, their "physical reality" can correspond to application software or to a kind of human instance performing the logic of that role. The sensors and actuators then correspond to human parts such as eyes, ears, arms, legs, etc. The agents in the hierarchy are defined to a relative granular level in order to detail and organize the application hierarchy in a standard and repeatable way and to understand the information dependencies.

Person roles and top level performer objects will be domain dependent. Person roles in the MIGVS domain include commander, gunner, driver, and various mission specialists. The commander can perform many roles and would be responsible for many tasks or desired trajectories as well as the overall mission. The mission agent would be assigned a commander person role. The gunner and driver are more specialized. The gunner can be related to Lethality as a top level performer object (i.e., it groups all the objects required to execute lethality, including an ALO hierarchy with a Gunner Task Agent the top of that hierarchy). By definition, a MCPS will have a driver person role linked to Mobility has a top level performer object. It will have an ALO hierarchy with a Driver Task Agent at the top.

As mentioned previously, for the logic of an agent to actually execute and interact with other agents in a system, an execution means is required in the form of a computation (e.g., brain, computer, neural net). Given the nature of a MCPS as described herein, a significant portion of the combined set of ALO computation should be assigned to a machine/computer. This computation can be described in terms of a "stack" as shown in Figure 34. Information flow can be understood to flow both vertically and horizontally through and across the stack. "Conceptually, the data moved horizontally and vertically is the same" (Shames and Sarrel 2015). As Shames and Sarrel indicate, the vertical flow can be thought to be a "transformation" of data or information. The logical interactions described thus far, can be thought to flow horizontally in the end-to-end execution of a behavior between interacting agents or like objects. This is the desired or objective behavior. However, the information flow actually goes through potentially many transformations up and down the stack before arriving in the proper form in its

destination. This occurs whether an agent has its own computational stack or shares one with the other agent. These transformations impact the timing of the end-to-end behavior as well as introducing the potential for errors, delays, etc. If standard or common information support software to support the ALO is desired, this can add further complications to the stack execution.

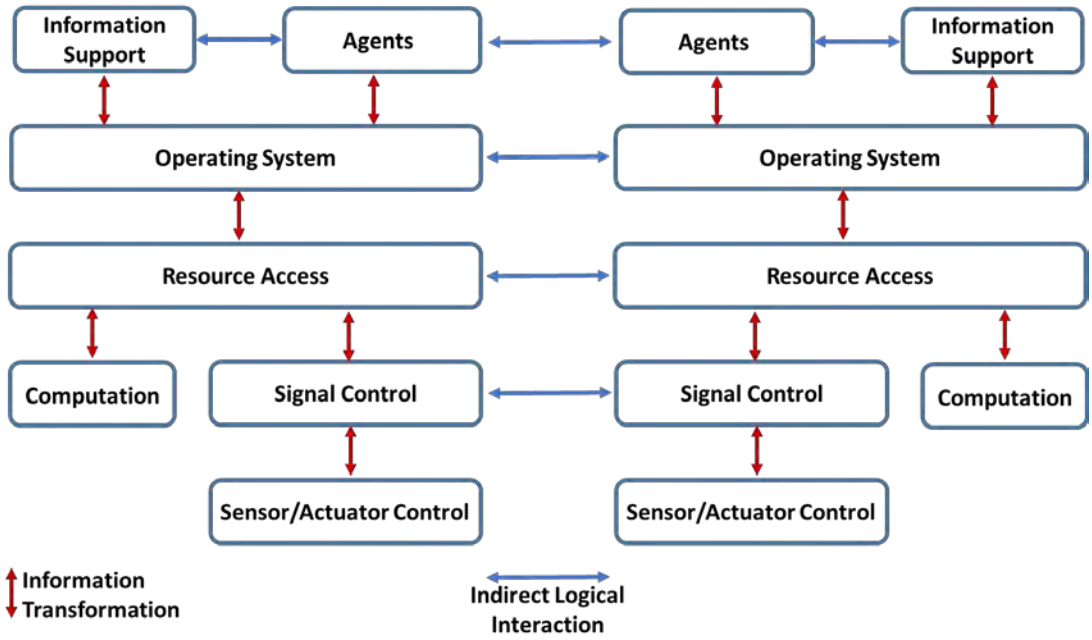

Figure 34. An Agent Computational Stack

A solution independent model would not require much detail relative to how specifically the vertical transformations happen. It would require knowledge and understanding that the horizontal flow or indirect logical interactions must be supported by the vertical transformations and they do not occur instantaneously, without cost or without some failure. The ALOs could be thought to "logically" execute on a single stack and its physical realization as a single assembly or an assembly for each ALO or something in between, would be part of the trade space required to finalize a concept design. The key would be to understand the vertical transformation MOPs and their relation to the horizontal MOPs and MOEs. By implication, the ALO hierarchy is also a

computation hierarchy linking eventually to the physical-mechanical behavior of sensors and actuators.

6. System Behavior Thread

A *system behavior thread* is defined as a singular horizontal path of an event, input, internal behavior logic, output, and then effect. An event was previously defined as a change of state in the external context that results in a system input. An effect was previously defined as a change of state in the external context caused by the system output. An *input* is defined as the system import of energy, material or signal/information and *output* is defined by the system export of energy, material or signal/information. The internal behavior logic can be as simple as the behavior of a single logical object or as complicated as a combinatory set of object behaviors with many object-to-object interactions. The *system logical behavior* is defined as the unconstrained combination of all threads. Agent logical object behavior and their interactions comprise the system intelligent logical behavior and is the combination of all intelligent behavior threads.

A system behavior thread is illustrated in Figure 35 for a single task ALO and below. The task agent receives a command from the Mission ALO. One or more detection agents detect events as state change in the external world or context. A single intelligent control agent's coordinated response through multiple controllers of actuators generates an effect or desired state change of the world. This event-effect "control loop" can repeat as needed until the desired effect is achieved. As defined previously, agents are just one type of system object and agent interactions just one type of interaction. ALOs interact with control system objects and they in turn with sensors and actuator objects. From a hierarchical behavior control standpoint, agent interactions to sensor/actuator control have been defined previously to be indirect logical interactions. The control system interactions with sensors and actuators take the form of signal energy. The interactions between the sensors/actuators with the world can take the form of energy or material, though only energy is shown for simplicity.

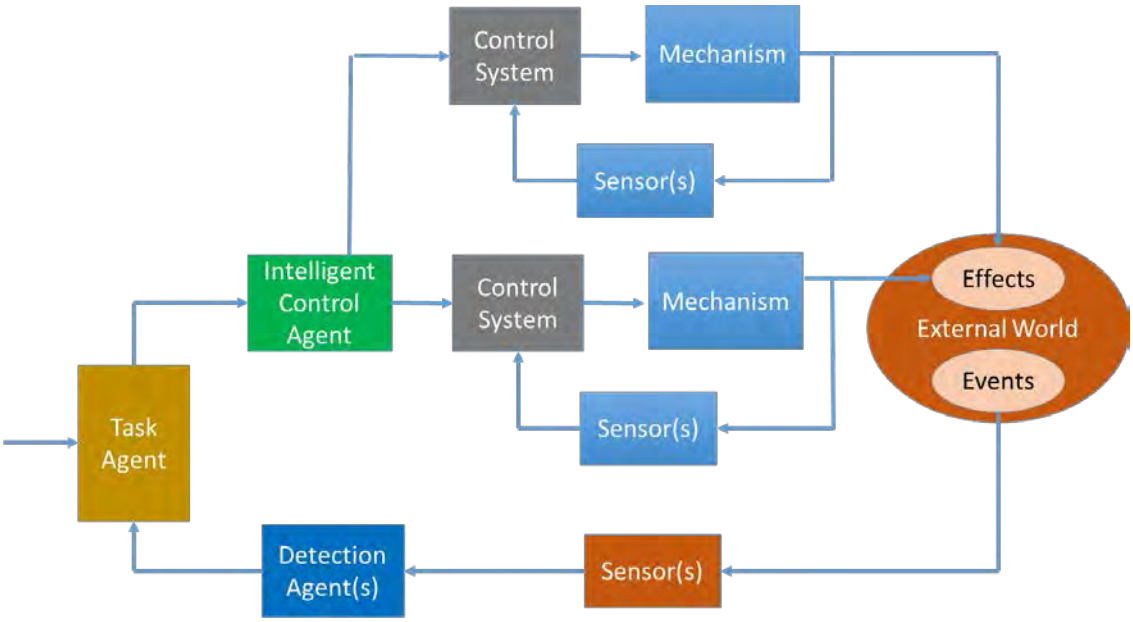

Figure 35. System Behavior Thread

The behavior thread then is that complete horizontal flow of indirect logical interactions to signals to events/effects and back again. This is supported at the upper levels by the vertical (Shames and Sarrel 2015) flow described previously. Just as the vertical flow represents the transformation of information, the horizontal flow represents its distribution. The concepts of transformation and distribution can be extended to energy and even material interactions. Not shown for simplicity is the control of the sensor. The detection agent would execute control commands as needed to get the information in a desired condition.

In summary, an ALO as application logic is similar to an application object as practiced in software OOAD and can be compared and contrasted, as shown in Table 3.

Table 3. Software Object and ALO Comparison

Comparative Item	Software Object (Booch et al. 2007)	ALO
1	A function is "an input/output mapping as a result of some objects behavior.	A task or intelligent function is an event/effect world mapping as a result of some ALOs intelligent behavior
2	An object "has state, behavior, and identity."	An ALO has state, intelligent behavior, and identity.
3	State is "the cumulative results of the behavior of an object"	State is the desired results or goals of the ALO's behavior relative to its current understanding of the world.
4	Behavior is "how an object acts and reacts in terms of state change and message passing"	Intelligent behavior is how an ALO acts and reacts in terms of state change and message passing relative to its goal state
5	An operation is "some work that an object performs on another in order to elicit a reaction …. message, method and operation are used interchangeably."	Message passing consists of commands and percepts and do not invoke a method, imply a transfer or singular thread of control, or provide a direct reaction to the sender.
6	State space is "an enumeration of all possible states of an object"	State space is an enumeration of all possible world entity attribute values of an ALOs world state model.
7	"The role of an object denotes the selection of a set of behaviors"	The role of an ALO denotes a unique set of intelligent behaviors that form its identity.

B. SOLUTION INDEPENDENT MCPS OBJECT ORIENTED BEHAVIOR CONCEPT DESIGN

To sufficiently capture the complexity of the solution independent objects, behavior logic, attributes, and interactions, as well as their integration into behavior concept design, a modeling language is needed. No modeling language provides the

syntax and semantics to directly model agents, components or physical object abstractions, physical and computational behavior, and data. SysML is selected as the modeling language that can best be leveraged. This is not meant as a modeling tutorial or to be an expert application or use of SysML. Key to its use here are the following features:

(1) It is the most commonly used modeling language for systems engineering. It is a descriptive visual language that supports certain object oriented techniques.

(2) The concept of a block provides the flexibility to model any structural type, to include complex data structures and complex context. It supports structural relationships such aggregation/composition, generalization/specialization and inheritance.

(3) It can model a variety of connector and interaction types. Proxy ports can be used to represent connector abstractions and then typed by interface blocks. It can be used to represent both horizontal and vertical interactions.

(4) General and descriptive modeling behavior techniques that can be incorporated into blocks or objects. It will support object oriented decomposition from various class types and then definition of behavior via leaf level classes or objects.

1. MCPS Domain Structural Model Concepts

As shown in the MCPS Architecture DM2 in Figure 27 and with consideration of comparison to the DODAF Architecture DM2 of Figure 4, the world can be viewed in terms of physical entities or objects and their interactions, as shown in Figure 36. This world is modeled structurally as having to two entity types, resource and context. A driver is one type of payload that has a person role. An MCPS by definition will have a driver person role. The block “Physical” is introduced here as an abstraction for material and energy so as not to bias the technological solution that uses the operating resource. Material can also be a type of payload. From a system perspective, an organization is part of the context. From a MCPS perspective, only organizations that it interacts with are of interest. Friendly force is a type of organization that an MCPS interacts with during operation.

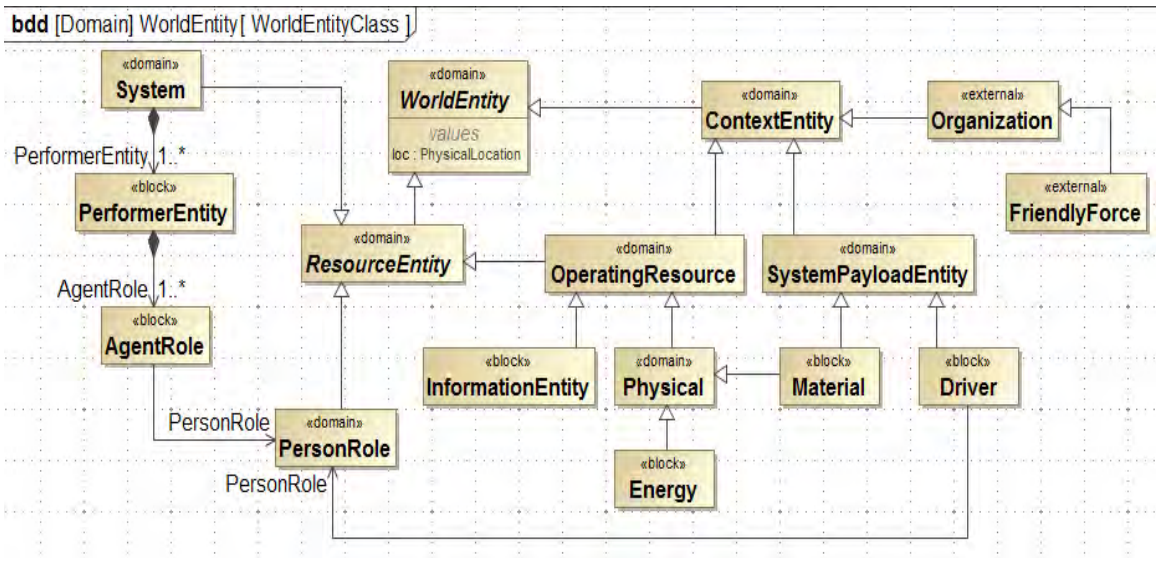

Figure 36. Types of World Entities

System is a type of resource as defined in the DoDAF DM2. It consists or is composed of one or more performer entities. Performer entities have one or more agent roles which also have a person role. The SysML notation indicates the "is a type of" relation with closed clear arrow. The "is a part of" relation is indicated by an open arrow with a diamond block on the other end. In SysML this also implies ownership, though not necessarily physical containment. The arrow without the diamond represents a reference compositional relationship but not ownership. Since much of the aggregation addressed in this approach is logical, the reference relationship will be used since it is not clear what logical ownership implies relative to physical ownership. The blocks represent entities that are abstractions of something physical. Information, similarly to software, can be thought to have a physical realization. A person role can also be considered to have physical realization. It could be an instance of a person executing a certain set of behavior or the making realization of that set of behavior.

a. World State and World State Model

From a MCPS system centric point of view of the world, the concept of a resource generally is not a particular useful modeling concept. A system is a resource per the DODAF DM2, but also uses resources. From a system centric standpoint, the world can

viewed in terms of two top level entities, the system itself and the context as shown in Figure 37. As mentioned before, the system has one or more performer entities which have one or more agent roles. The context is composed of multiple entity types. Two example types discussed so far are the system store and friendly force. The system store is an overarching concept for context objects that can be contained within the system boundary: person role, operating resource and payload. The friendly force represents other systems or organizations that the system interacts with symbiotically; that is, entities with which the system interacts to achieve some higher level objectives beyond what the system can achieve singularly. The generalized concept of an organization is not needed in this approach.

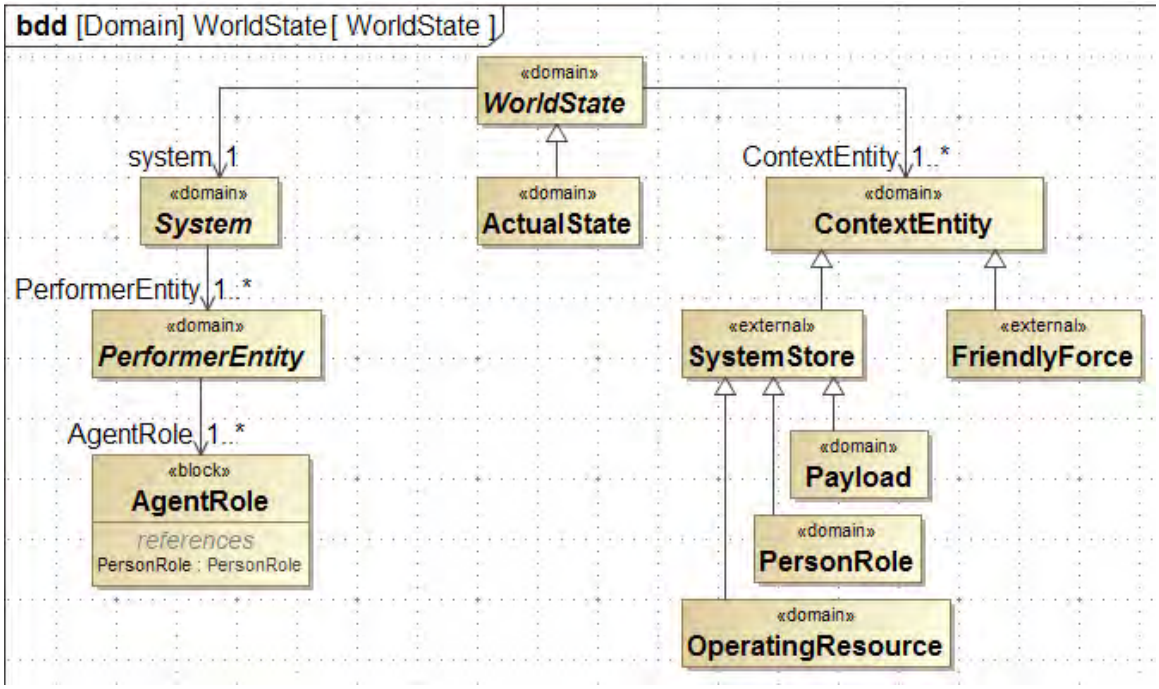

Figure 37. World State

Each of these objects or entities have attributes or state variables. The value of an object's attributes at a given point in time reflects the objects state. The set of all possible values is the state space. Collectively all the world objects and their attributes comprise the *world state*. The actual state of the world is the ground truth state variable values at a given point in time. From the standpoint of systems engineering, the system is designed

as needed to appropriately interact with the context. From an agent standpoint, this world state needs to be tracked and understood. This representation of the world is defined as the *world state model* and the current value of the object attributes or state variables is the belief state.

b. Context Class

In addition to system store and friendly force, there are other context entities or classes that are relevant to any MCPS and that further define the domain. These are shown in Figure 38 arranged in a composition. A common domain context attributes could be arranged in a class inheritance model. The inheritance class model would identify a sort of "pick list" of top level classes for the composition of the context in a given system application. These classes are defined as follows

(1) System Connected—context object that can be stored, housed and/or used within or at the system boundary

(2) Friendly Force—a context object of another system or organizations that the system interacts with in some symbiotic fashion (e.g., two systems cooperating to achieve some higher level objectives beyond what the system can achieve singularly).

(3) Terrain—by the definition, the mobility aspect of an MCPS implies that the system traverses some sort of terrain (air, ground, water, etc.). This terrain can be decomposed or structure into a set of classes with objects and attributes.

(4) Meteorology and Weather—for many MCPS, particularly those that traverse the surface of a terrain, the meteorology or weather are a distinct aspect of the context that can impact the behavior the system, particularly the intelligent aspects of the system that rely on sensor understanding of the context.

(5) Civil—whether expressly designed for civil use or not, most MCPS will likely have to consider or interact with civil entities. These objects include pedestrians, non-friendly force vehicle systems, cultural features and traffic management components.

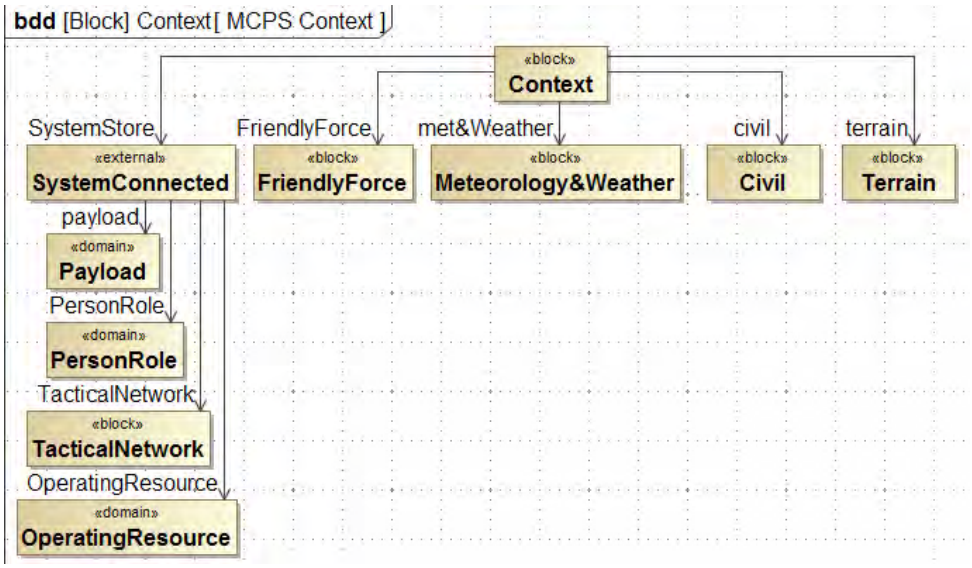

Figure 38. MCPS Context Classes

Whether arranged as an inheritance or composition model, the context has to be decomposed to some leaf level where objects with attributes can be instanced. System store is an example of the next level of decomposition, but is still not at the leaf level. A particular decomposition can be better focused for a given MCPS sub-domain (e.g., MIGVS). Also, the second level composition of the context can be extended for a given sub-domain. For example, top level concepts for a MIGVS context include military base and enemy or threat in addition to what is shown in Figure 38. As indicated previously, terrain can include roads which can include lanes as shown in the example classification of Figure 39. A road is a type of improved surface which a type of a terrain "thing" and has at least one notion of a lane which can have lane markings. These lane markings then have a set of attributes which can impact how the system behaves. Conversely the terrain for a given MCPS can be viewed as being composed of a set of improved and unimproved surfaces.

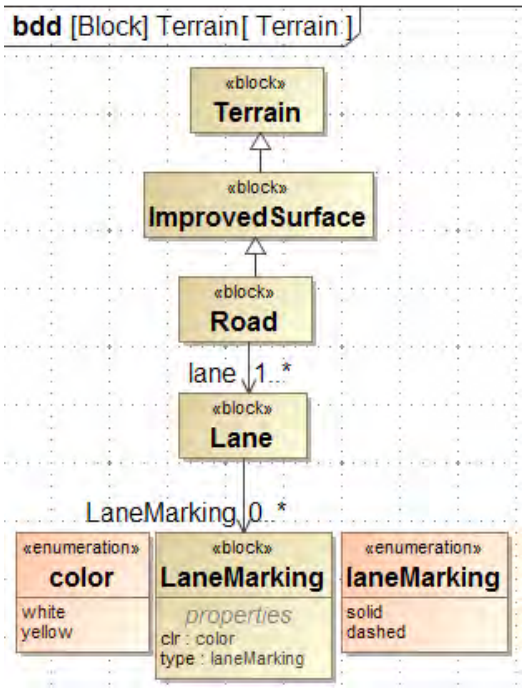

Figure 39. A Terrain Decomposition

c. System and System Performer Classes

Systems are composed of system performer objects. As with the context, these system performer objects can also be considered domain MCPS classes. These are the top level performer classes that are in turn composed of sub-performer classes or objects which in turn can be further decomposed. When considering MCPS as a domain, these are classes that organize the domain and may have like properties and attributes. When considering a specific MCPS, these form logical compositions or aggregations of those lower level types. Common MCPS performer classes are defined as follows:

(1) Mobility—a set of performer objects that provide the prime force that enable MCPS movement and the ability to control and support that movement as required.

(2) Tactical Command Control and Communications (TC3)—as indicated previously in the context discussion, it is rare for any MCPS to operate without some connection and purposeful interaction with other systems. These are the set of performer objects that enable and MCPS to communicate and exchange information with those systems.

(3) System Command Control and Autonomics (SC2A)—regardless how an MCPS is physically assembled, it can be viewed logically as a set of autonomic objects that transform and distribute power and that control signals and information as required to support other performer objects. This autonomic system also has objects that manages the transformation and distribution. Since the mission agent requires control of this autonomic system to best execute, it is included here as well.

(4) Structure—the system performer classes previously described can be decomposed to some component level. To form a system, they must physically assembled in some fashion. These are the endo and exoskeleton objects of the MCPS. These objects can be considered to structurally interact with each other and the components that they support.

(5) Mission or Special—in addition to the common or standard MCPS domain classes described above, a given sub-domain of MCPS can expect to see some specialized or unique mission purpose that can also be logically related as a set of performer objects.

The intelligent and relatively more active performer objects that are the focus of this research have a common set of object types: external sensor/actuators, direct control of sensors/actuators and agent logical objects as discussed previously. This is shown as an MCPS domain view in Figure 40. Each of these performer classes have operating resources subject to capacity constraints and performance attributes subject to design constraints. The intelligent control pattern, including task, detection and intelligent control agents, is shown for Mobility but can be similarly elaborated for each top level performer class. Like the context, these domain classes form the "pick list" from which to compose or aggregate objects into systems. Further elaboration can only reasonably be done within a given sub-domain (e.g., MIGVS). The objective again is to get to the specific leaf level classes and meaningful instances of physical objects within definitive attributes.

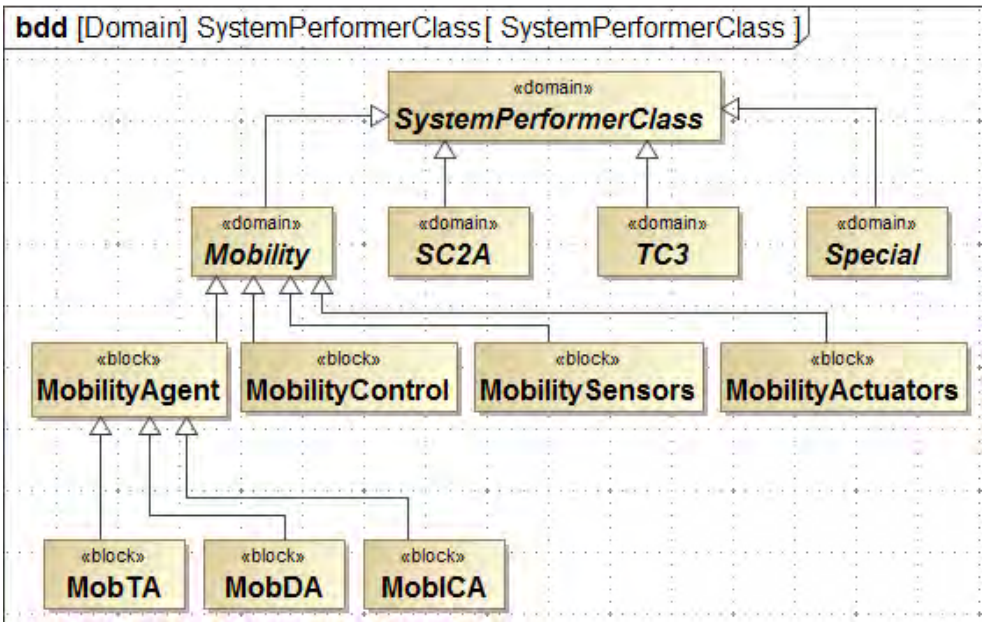

Figure 40. MCPS Domain Performer Classes

d. Assigned Mission

An assigned mission is an externally supplied objective or purpose and is modeled as shown in Figure 41. As indicated previously, trajectories include world entities which define a state space of interest and goals which define desired values relative that state space. As before, there are three types of trajectories: mission effect, discipline and exception handling and each desired trajectory has an end state goal that is to be achieved or maintained. Military operations generally have a significant planning phase where each participating system receive detailed steps or a priori objectives that can be synchronized with other systems. The system needs to distinguish between planning to achieve a goal versus executing an action that achieves the goal. An Army MIGVS mission includes other phases (phC): preparation, execution, and refit. An Assigned Mission have an ordered set of tasks (taskOrd), tasks have an ordered set of desired trajectories (trajOrdN), and desired trajectories have an order set of goals (goalOrdN). A *mission state* is defined as the last task and trajectory activated and the goal state of any other desired trajectories still executing.

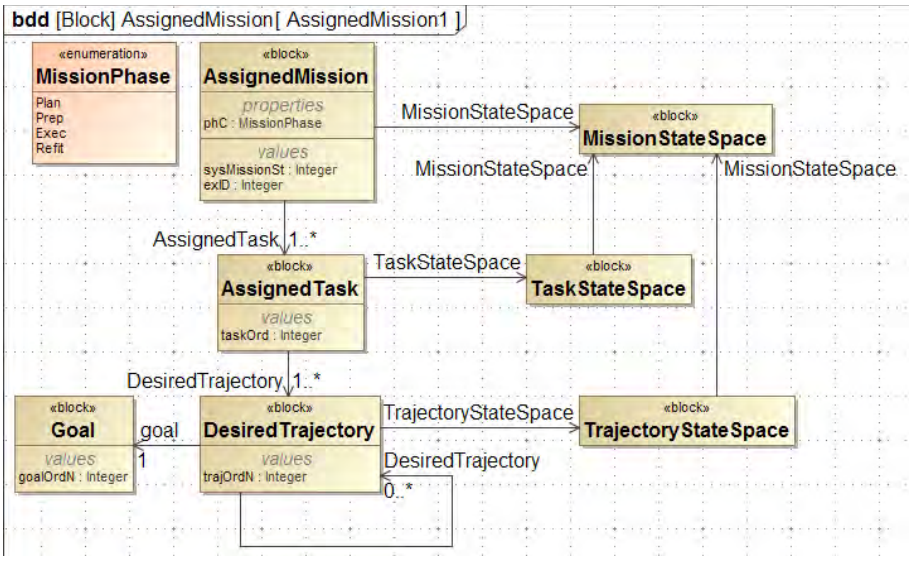

Figure 41. Assigned Mission

Goals then are satisfied not only when they achieve or maintain the correct state (satSMOP) relative to the correct time (satTMOP) per the goal specification of Table 2, but also when they execute in the correct phase and correct order. The mission state will reflect a current phase (phC) and a current desired trajectory order (ordC) that can be compared to the phase (phase) and the order number (ordN) of the goal as planned. Goal satisfaction (isSat) can be represented in Boolean expressions as follows:

isSat = {satSMOP} and {satTMOP} and {isActive}

satSMOP = {current state <= (state target +/- state tolerance)}

satTMOP = {current time <= (time target +/- time tolerance)}

isActive = {phC = goalPh} and {ordC = ordN}

Mathematical expressions in SysML are captured using constraint blocks. As shown in Figure 42, constraint blocks are used to capture the expression above and are part of a goal. Achieve and maintain goals have some variation in state and time expression, so goal is further typed accordingly. Also, since complex state is difficult to express mathematically, particularly in a SysML constraint equation, high level representation of that state may be used along with value expressions as shown in Figure 42. For example, given a vehicle following goal target of 40 KPH with a tolerance of +/-

2 KPH, can be equivalently expressed as a target value of one with a tolerance of +/- 0.05. For a knowledge state situation goal, the state measure can be expressed as a difference between the set of all possible variables and the values of actual state variables.

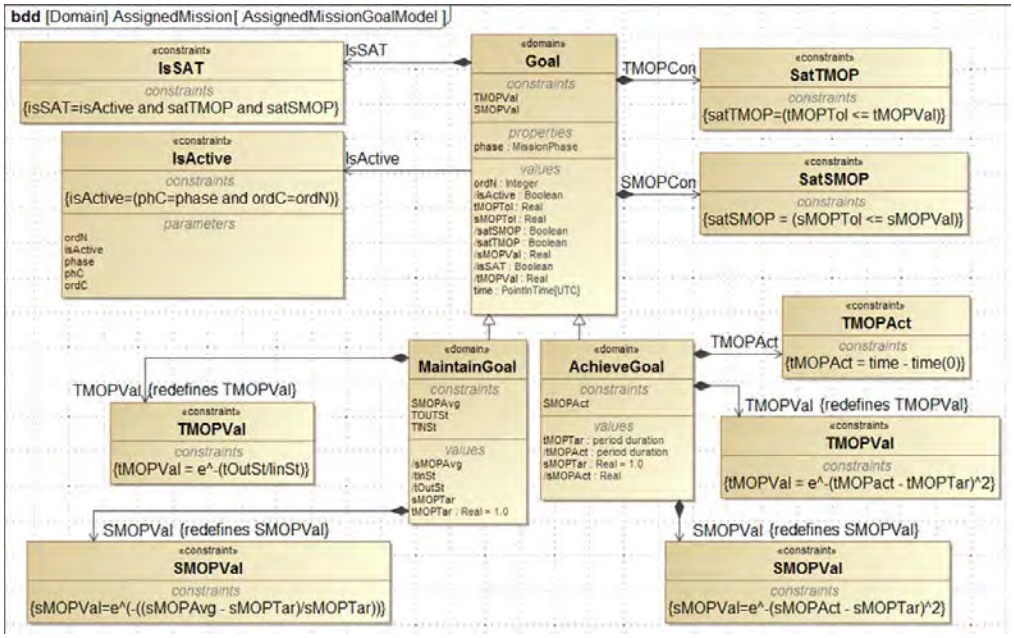

Figure 42. Goal Constrains

Specific goals through the mission and desired trajectory hierarchy are typed as achieve or maintain goals, at which point they will inherit the appropriate constraint block equations. Since the value equations are notional and time targets and tolerances are straightforward, the focus of the modeling and goal constraint equations is on the target and actual or current state. Each state expression is unique to given desired trajectory and goal and can have many variables. Additionally, for maintain goals, the current state needs to be assessed throughout execution and will be an in-progress state measure, such as an average or accumulation. A state average measure is used as the default for a maintain goal state equation in Figure 42. Since the state measure needs to be re-defined or retyped for each unique goal regardless, it can be re-defined to the appropriate in-progress measure at that time.

e. Agent Logical Object Model

A general pattern for an agent logical object's structure is shown in Figure 43. The agent is composed of an overarching behavior and a world model. The agent's behavior acts on the data stored in its world model. The world model includes Agent Knowledge and a World State Model (WSM). Knowledge includes declarative and procedural knowledge, particularly knowledge about standards of behavior. The WSM includes an Assigned WSM (ASWM) and a Derived WSM (DWSM). The DWSM is composed of the subordinate agent AWSMs. The agent's behavior then generates commands to subordinate agents and percepts to superior agents as required given the current or belief state of the world and its goals. Each agent follows the same general pattern as indicated in Figure 43.

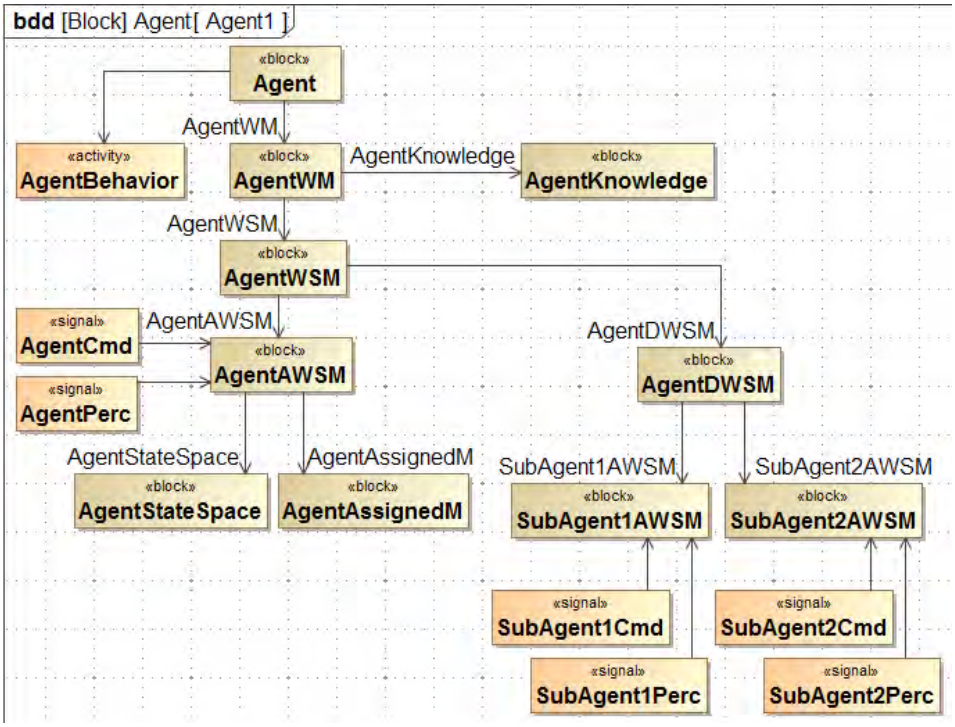

Figure 43. Agent Logical Object Structural Pattern

The top level agent structure for a given MCPS consists of the mission agent and some number of task agents. An example system's top level agent composition is shown in Figure 44 using a SysML Block Definition Diagram (BDD). Each of these agent types

represents a more specific person role and has the pattern shown in Figure 43. The mission agent is one role that a vehicle commander performs within an MIGVS for instance. Each task agent can decompose in a similar fashion to include an intelligent control agent and some number of detection agents within a BDD. They in turn also decompose into some number of sensors and controllers. A control hierarchy can thereby be fully elaborated from a mission agent at the top down through some set of sensors and controllers at the bottom.

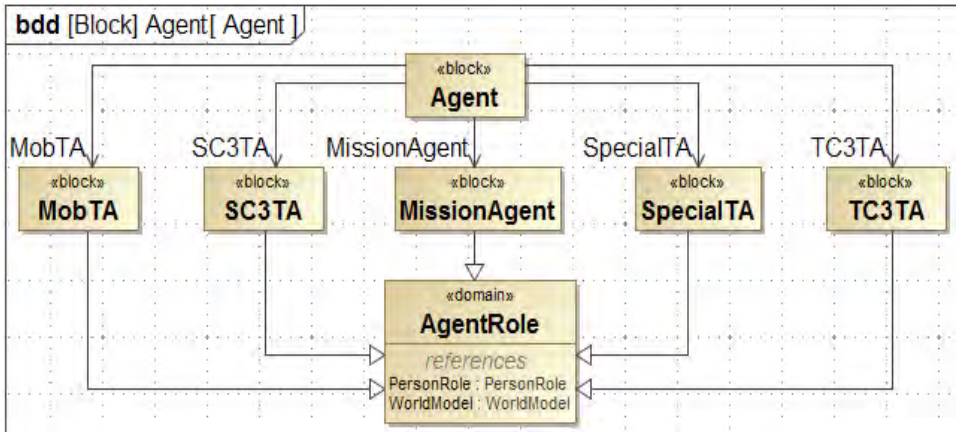

Figure 44. Example Mission and Task Agent Structure

Agents then interact via commands and percepts in a hierarchical pattern as shown in a SysML Internal Block Diagram (IBD) in Figure 45 for the mission and task agents. The "internal" in this case does not represent any physical containment. The commands and percepts are represented as signals that have directional flow between SysML proxy ports. Both the signals and the ports can be thought as some logical component (e.g., message and interface, respectively). The flow of information can go from the mission agent through tasks agents, ICA and detection agents, to controllers/sensors and back again. This flow of information represents indirect or horizontal logical flow as defined previously. Each component in the hierarchy has the internal behavior to track its world state of interest, store it in a world state model, and take appropriate action given changes in the world state based on the information flow.

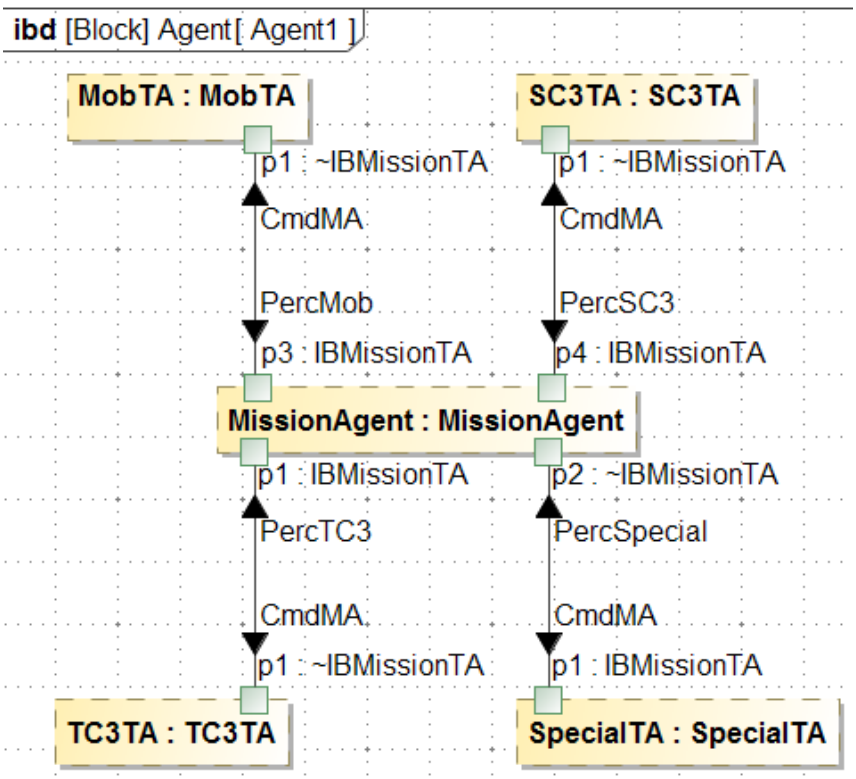

Figure 45. Mission and Task Agent Hierarchy

2. Solution Independent Behavior and Logical Concept Design

The structural concepts just presented are used to logically compose a system and its context in terms of objects. The system's behavior can then be defined as the interactions of those objects. These object interactions have both "horizontal" and "vertical" aspects to consider. The internal behavior of a given ALO acting on the goals of its assigned and derived mission drive the horizontal interactions.

a. MCPS Logical Composition

As shown in Figure 46, from a system standpoint, the world is composed of two major entities: itself and everything else with which it interacts (i.e., its relevant context). Both the system and context can be "logically decomposed." As identified previously, a general model of an MCPS can be defined in terms of the top level performer entities of Figure 40. The top level performer entities can be further decomposed to some leaf level object that has behavior. The decomposition is best done in the context of a given sub-

domain. Note again that these leaf level objects are abstractions of physical things. Also note, that the top level performer objects are then logical collections of these physical abstractions that can have aggregate properties and behavior. The context entities can also be decomposed to objects or leaf level classes as required so as to define the world state space together with the system objects.

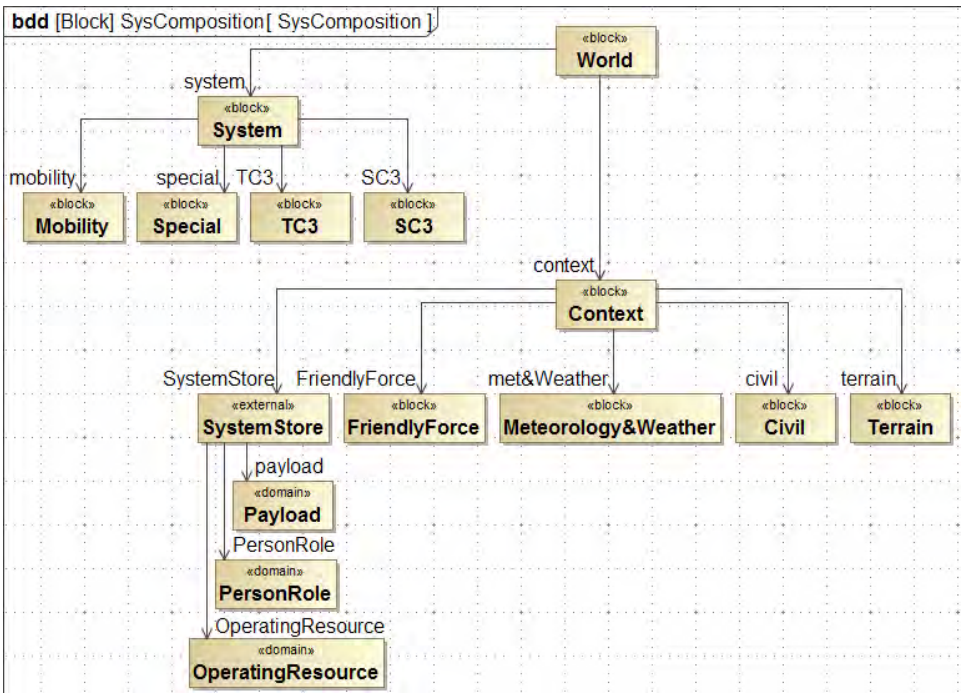

Figure 46. System and World Composition

An example mobility logical decomposition is shown in Figure 47. With the exception of mobility agents and the mobility intelligent sensors, these entities represent actuator-controller pairs. For example, brake includes the brake itself and the brake controller. They are not distinguished for simplicity of presentation. The controllers also include any feedback sensors. The intelligent sensors are sensors that detect human-like intelligence and/or context understanding. They are part of the event-effect control loop shown in Figure 35 and would be paired with a detection agent based on what is being detected. For example, a stability sensor/stability detection or a lane sensor or lane detection agent.

The blocks should be solution independent as much as practicable in the initial conceptualization, though it is not likely to be system or sub-domain independent. A MCPS, or a least a ground vehicle system, can be thought to have an engine even it is largely composed of a battery pack. It should also have a transmission whether it is a set of mechanical gears and linkages or electric motors and linkages. The engine and transmission controllers are more problematic. However, the control of power or power draw to an electric power can be considered logically distinct from control of its direction no matter how likely they are to be combined in a physical solution. Secondary power generation supports specific types of loads. Examples solutions for ground vehicles would be an alternator or DC-DC converter to support electrical/electronic loads.

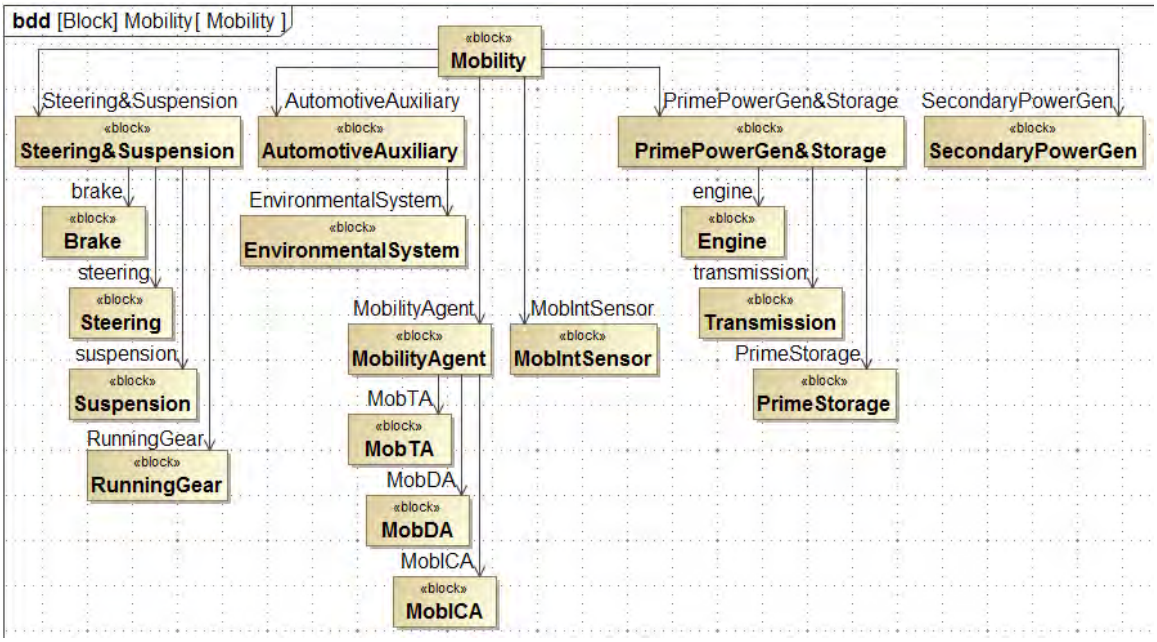

Figure 47. Example Mobility Logical Decomposition

b. Conceptual Data Model

All the system entities can be elaborated similarly to Figure 47. As indicated previously, the elaboration of mission, trajectories and goals results in the elaboration of the world entities, both system and context. The full elaboration of both system and context will define the entire world and world state of interest to the system. However,

each ALO world state model may duplicate entities relative to other ALOs. Given the potential data intensiveness of this approach and given the need to understand the full context relative to the system to support system analysis, a conceptual data model is warranted. A conceptual data model provides a means to directly reason about the data in the system. Though often used relative to data base design and data flow diagrams, it can also be a convenient view into the data that augments or aids an object or object-like design. A singular view into the data provides a visual aid that can promote efficiency and interoperability for ALO world state development and update. Dividing the entire world of interest into a set of concepts makes the reasoning about it tractable. It is akin to decomposing the external world into a set of features (Poole and Mackworth 2010) and is certainly a far more practical approach than reasoning about the world directly in terms of states.

c. Agent Logical Object Interactions and Behavior

The logical agent interaction of Figure 45 can be expanded with the mobility agents introduced in Figure 47. This is shown integrated in Figure 47 to emphasize the hierarchical agent interactions of Figure 33. Each agent receives commands from a superior agent and sends percepts back. Conversely, each agents sends commands to subordinate agents and receives percepts back. Each task agent could be similarly expanded. Each would have only one intelligent control agent and many detection agents. The intelligent control agents communicate with the controllers of the control-actuator pairs identified in Figure 48. Each detection agent would communicate with one intelligent detection sensor for control and data. The command and percept extend to these connections as well.

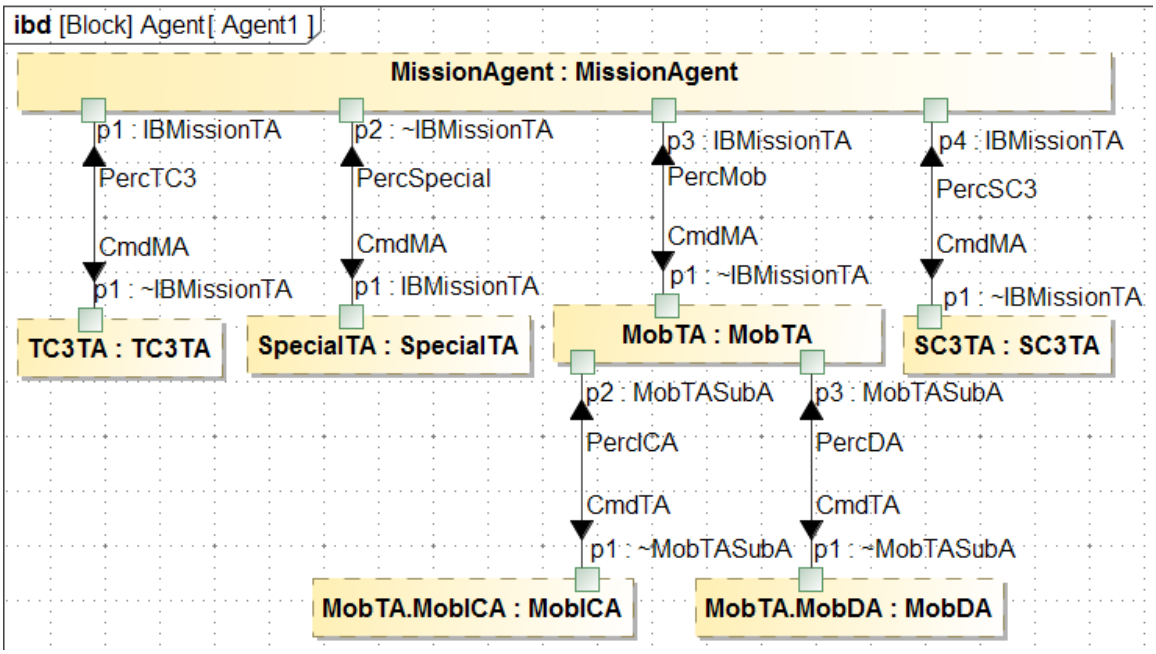

Figure 48. ALO Hierarchical Interactions

The ALOs are leaf level classes that have behavior when physically instantiated as objects. This behavior is shown in Figure 49 for the mission agent. However, each ALO follows the same behavior pattern. The command/percept parameter nodes of Figure 48 on carried by the command/percept signals shown in Figure 47. A command is received from some agent superior to the ALO in the agent hierarchy. This command is stored in the assigned mission state model (ASM) which is part of the ALO world state model shown in Figure 43. The command is analyzed to see what behavior is required to achieve the assigned mission. The results of this are derived mission state models (DSMs) and commands sent to subordinate ALOs in the agent hierarchy. The subordinate agents send back percepts to the ALO which then updates the DSMs, the ASM and a percept is sent to the superior agent. The cycle repeats itself as required and can extend down to the controllers of the sensors and actuators. After the initial command, the ALO will access what is currently stored in ASM and reconcile as required with derived state and commands.

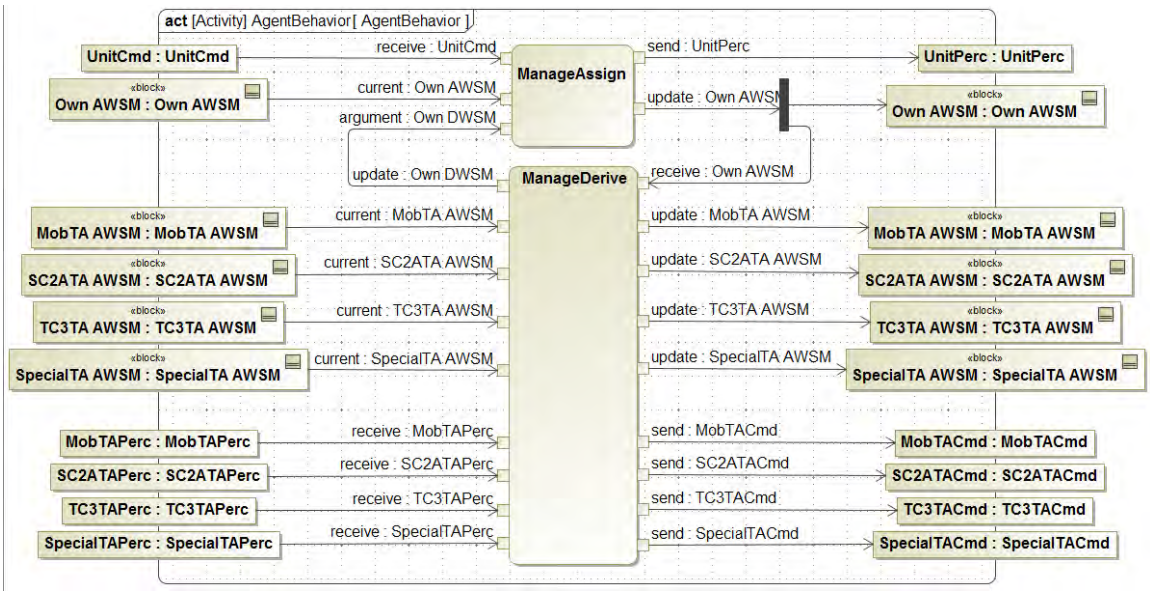

Figure 49. Mission Agent Behavior

The assigned and derived state models include goals, goal states and current values about the real-world state space of interest to the ALO. The goals may include MOEs/MOPs appropriate to the assigned missions and trajectories. The derivation of missions and trajectories would then include MOE/MOPs required to support the assigned mission. Note that the ALO behavior is pretty basic at this stage of concept design (i.e., bring information in and move information out). The behavior is of course a much more challenging design problem if it is to be realized by computational technology rather than a human, to include the definition of a priori declarative and procedural knowledge required to behavior with a certain expertise. However, in either case, the behavior is limited by the information it receives and ultimately what the system can sense about the real world. For this stage of concept design, the behavior is focused on what information is needed and therefore what has to be sensed.

d. Horizontal Behavior Logic

As defined previously, the system behavior logic is the sum of all system behavior threads. These system behavior threads represent horizontal logic of indirect interactions that can trace from event to effect. A notional example of this horizontal behavior logic is shown in Figure 50 using the previous example of mobility. The mobility task (MobTA)

agent receives commands (CmdMA) and sends percepts (PercMob) to the mission agent and stores current state in the assigned mission state model per the ALO behavior pattern of Figure 49. It then derives mission for the intelligent control agent (MobICA) and in this case, one detection agent (MobDA). The MobTA sends the mission and any updates by command to MobICA and MobDA and receives status back via percepts. The MobICA derives and commands the control signals it needs to meet its assigned mission and sends signals to engine, transmission, steering, and brake control. The MobDA commands the road sensor as required. Both the MobDA and MobTA receive percepts back. From this point on, the interactions take the form of direct energy or signal flow to include interaction with the environment.

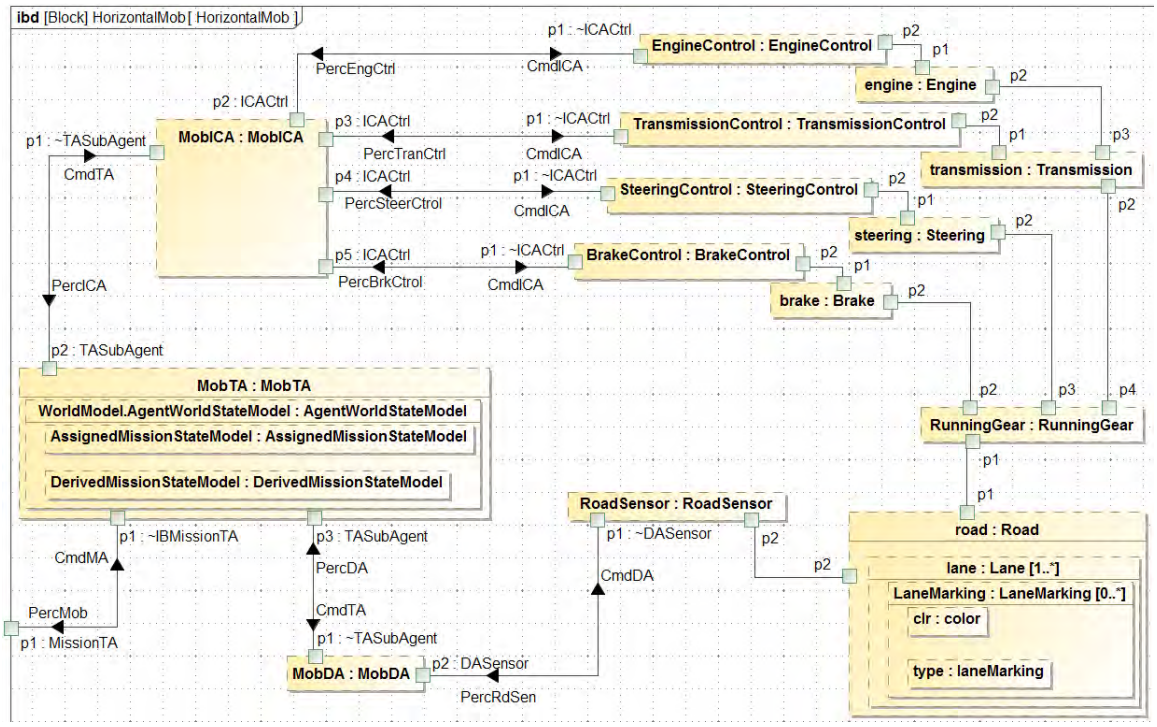

Figure 50. Mobility Horizontal Behavior Thread

Only the running gear and the road sensor directly interact with the context. The MobICA and MobDA ALOs provide the respective logical interpretations of those interactions to the MobTA. Only the ALOs can detect the occurrence of an event and determine if a desired effect has been achieved. The MobTA decides on the appropriate

course of action based on the context and system status as compared to the assigned goals and then issues any required course correction commands to the MobICA. The context and system status reflect the current values of the world state space and the goals reflect the desired values. Any MOEs/MOPs are part of the goals and are decomposed through the ALO assigned and derived mission and can be linked to control system, sensor and actuator MOPs. The MOE/MOP dependencies can be defined by parametric diagrams in SysML and constraint equations. A full mobility mission would require more knowledge of the context, a wider state space, and therefore more mobility detection agents and intelligent sensors.

e. Vertical Behavior Logic

As defined previously, the ALOs require the support of a computational stack, either machine or human. This computational stack enables the indirect logical actions between agents through signal, data, and information transformations. The main purpose of the SC3 performer entity is to provide the autonomic control of the system, to include the computational stack, required to support the mission ALO and the indirect logical interactions of all ALOs. The computational stack of Figure 34 is shown in SysML in Figure 51. The key elements of the stack are listed below.

(1) Information Support—this is more of a development reality than a necessary component of the stack. Rather than have unique information components for handling similar information (e.g., maps), common information support components will likely be utilized and interface as another application.

(2) Operating Environment—provides logical abstraction service (e.g., data management), between applications and the operating system.

(3) Operating System—provides logical abstraction services to the computer for scheduling tasks, managing memory, etc.

(4) Resource Access—provides direct access to computational hardware and signal devices (e.g., device drivers).

(5) Computation—computational hardware

(6) Signal Control and Distribution—components that transform signals from one form to another to support distribution and physical interface

The purpose of the computational stack is not meant to provide a standard or precise definition, but understand the flow of logic and its constraints and therefore the impact on horizontal behavior thread performance. The flow of information does not go directly from one ALO to another as it might appear Figure 51, but takes one too many paths up and down the stack and goes through many transformations through different forms of information, data and signals. The constraints of this transformation and distribution are the performance attributes of the vertical stack components. These attributes include latency, throughput, various computational measures, various information or interoperability measures, and measures of the reliability or health of the components themselves.

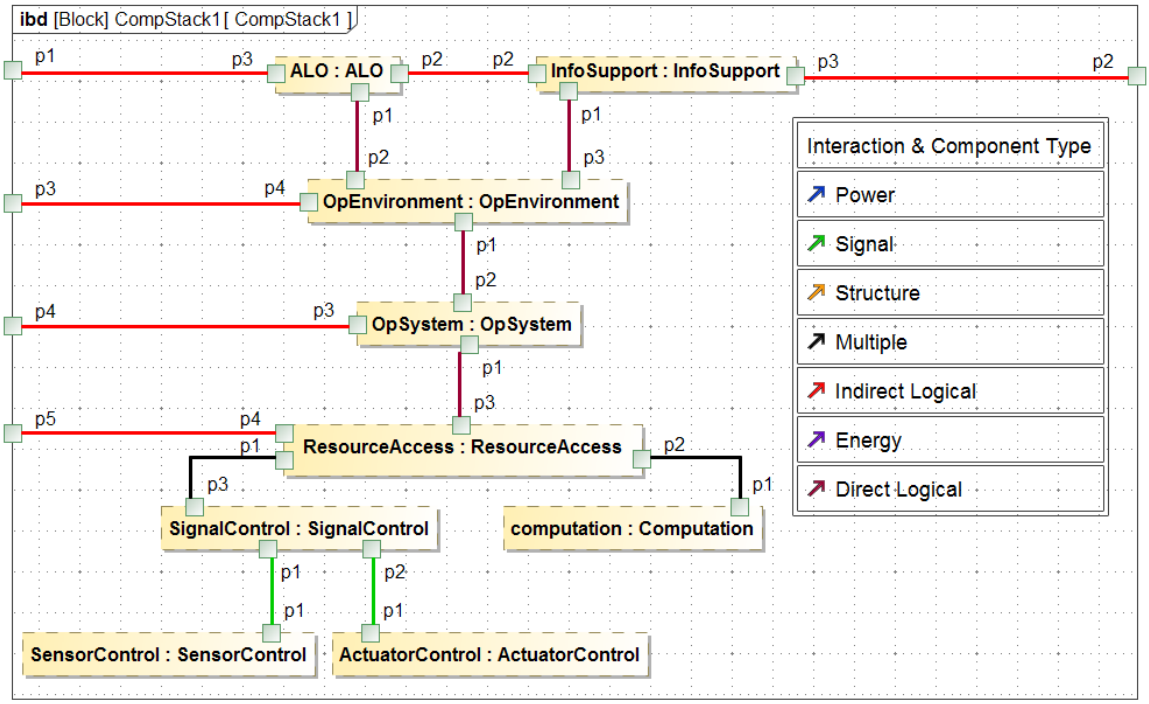

Figure 51. Computational Stack

Quantification of these MOPs cannot be defined until technology and architecture selection as well as identification of human operators if any, but the linkage of dependencies can be established as well as reasonable bounds on the MOPs. The architecture can range from a single stack for ALOs or a single stack for each ALOs.

These architecture extremes drive consideration of concurrency and have a different impacts on the MOPs, such as signal delays versus operating system access delays. In addition to the computation stack, other resource stacks for structure or power, particularly electrical power, could be similarly defined if considered critical for concept design. Structure is defined in this approach as its own unique performer entity, but electrical power is considered part of the SC3 performer entity. SC3 thus includes the components that make up the MCPS autonomic system the potential for direct analogies and allocation to any needed human autonomic system.

The solution independent logic design is completed with the integration and iterative elaboration of these models just described. The object-oriented nature of the system classes should make a more iterative approach feasible (e.g., adding a new class does not break previous class definitions or their relationships). Also, the full definition and elaboration of all system objects, attributes, and their world relationships, is not necessarily needed to adequately inform the trade space. Certainly critical MOEs/MOPs along with their dependencies must be captured that drive trade space decisions of system architecture, technology solutions and any human operator allocation. However, since what is critical is not always known at the beginning of concept design, the object oriented system approach can provide a "whole" model of the system logic with varying fidelity based on its role and importance in the trade space analysis.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. PALLETIZED LOADING SYSTEM CONVOY FOLLOWER

The concepts just described and modeled are applied to a MIGVS as a case study. The MIGVS selected is the Palletized Loading System (PLS). To understand the development of an agent and object based PLS concept model, a perspective of the MIGVS as a domain is first presented. The MIGVS domain is presented in terms of a domain system model, domain context model, an overall approach to concept design and the PLS as a design reference system and convoy following as a design reference mission. The domain system and context models are used directly in the generation of the agent and object based concept model.

A. MIGVS DOMAIN

At a certain level of abstraction, all MIGVS have nearly common logic—shoot, move, communicate and survive; this is so particularly for combat MIGVS. They are distinct mainly in the performance and constraint attribution of the logic. This logic can be embodied in a set common system performer class abstractions as well as common functions. Many tactical vehicles however have specialized missions and capabilities, such as launching a bridge, conducting route clearance, or in the case being studied here, hauling supply. These can be added to the common logic as a set of specialized system performer classes.

1. MIGVS Domain System Class Reference Model

System performer objects for the MIGVS domain can be hierarchically organized consistent in part with the theory of *nearly decomposable systems* (Simon 1962), or arranged in hierarchical levels so that the interactions within a level are a "different order of magnitude" than the interactions between levels. As Simon explains, hierarchies are composed of interrelated subsystems which are also hierarchical in structure and can be further decomposed until some lowest "elementary" component is reached. Simon also emphasizes hierarchies that are based on "who interacts with whom," not on any necessary "spatial" relation. The "elementary" component in a software OOAD logical view can be defined as a snippet of code or a software "physical" part. This snippet of

code can be configuration item assembled into programs, executables, etc., which form the equivalent of a physical hierarchy for software. These are different hierarchies and may or may not bear similarities in structure. The hierarchy of a MIGVS logical structure follows the same principles, but its "elementary" components include more than software code. They include any components that can be instantiated from all the system objects and their sub-types.

In addition to the physical assembly and logical structures, other hierarchies could be considered relative to the same components. How power is generated, distributed and then consumed is a hierarchy, how structure is organized and support components could be another hierarchy. Each of these can also be described as a different kind of interaction. Any give hierarchy needs to be based on a type of interaction. For a logical hierarchy, two types of interaction need be considered: 1) the logical dynamic interactions that occur for a particular behavior or set of behaviors, and 2) and a non-dynamic interaction or long term interrelationship that occurs to achieve an overarching purpose. Thus, a *logical structural hierarchy* (LSH) is defined as a grouping of objects/components according to the preponderance of its logical interactions to achieving an overarching or common purpose. This hierarchy along with the other hierarchies mentioned are different views into the same components for a given system and represent a design vocabulary constraint.

A ground vehicle domain model (Adams and Washington 2012) was developed that attempts to define a logical structural hierarchy for all possible domain components, including cyber components such as software, control systems, etc. It was referred to as the Standard Product Classification Hierarchy. This MIGVS domain model has been adapted consistent with the objectives of this research. The first three levels of the adapted model are shown in Figure 52. The MIGVS domain model organizes the logical hierarchy into a pattern that can be utilized by any MIGVS project and forms a superset of all possible elementary components. As such, it provides an object/component view of all possible logic for a MIGVS. The intent is for the logical structure of any particular MIGVS to be a selective instantiation from this domain model. It has the potential to provide a standard logical view into a systems EBOM as the physical view standard of

MIL-STD-881C. The leaf level components should be exactly the same except that it can provide greater granularity of software, electronics and control systems components and does can be used prior to a physical assembly and integration schema being selected. It can be considered a decomposition bill of material (DBOM).

Figure 52. MIGVS Domain LSH

The MIGVS domain logical hierarchy in Figure 52 is organized into eight logical groupings or classes at the second hierarchical level. Though appearing functional, these are logical groupings of component abstractions or system objects. Each object at each level in turn has a hierarchy until arriving at some desired "leaf level." The leaf level can be relatively high for early system abstractions, such as initial concept design, or can extend to the part level equivalent to a full EBOM if desired. The organization again is based upon the dominant form of interaction required between objects at a given level to achieve the fundamental purpose associated with the object above them. There are a few

exceptions to this which will be explained. The second level logic is defined with discussion as follows:

a. Structure

The objects/components that provide primary and secondary structural support to all system objects/components as well as protection against the direct effects of the external environment. Component examples include turrets, hulls, cabs, electronic bays, hatches, and software "structural" components (e.g., files, executables). Components providing structural support have an interaction consisting of a balancing force or energy. Software structural components do not have interactions with the physical structural components, but contribute to the overarching purpose of providing structure. Structure does not currently have agents, but this is a current technological assumption and agents can be added when warranted. For example, embedding nanotechnology within the structure could perhaps change its properties in response to dynamic conditions or provide a behavior, such as health self-assessment.

b. Survivability

The objects/components that provide system-level protection against the direct and indirect effects of projectiles and explosives. Examples include armor, active protection systems, signature management systems, and fire extinguishers. Survivability currently does not have overarching agents to perform integrated survivability, again a technological assumption similar to structures. However, constituent survivability objects/components, such as active protection systems and autonomous or automatic fire extinguisher systems, can be modeled with an agent structure corresponding to roles.

c. System Command Control (C2) and Autonomics

SC2A is the set of objects/components that orchestrate mission and mission behaviors through the command and control of system resources, provide the underlying information/computation system resources, and that provide the behaviors required to maintain internal local situation awareness as well as certain actions as a consequence of that situation. A *mission behavior* is defined as any behavior that requires the support of

more than one tasks agent. This can include the overall mission, complex maneuvers such as hasty defense (Department of the Army [DA] 2012), and other relatively complex behaviors such as full spectrum operations and mission tasks (DA 2012). The mission agent as shown in Figure 45 is included here even though it has indirect logical interactions across the hierarchy. However, by definition it contributes to the SC2A overarching purpose. Also, it has a more or less continuing interaction with the SC2A task agent, so it is likely to have more interactions with it than any other task agent.

The behaviors represented by the SC2A agents are historically the responsibility of the vehicle commander. These behaviors include: leading the system mission, leading the crew collective tasks, managing system resources through the mission, and maintaining internal situational awareness. The mission, SC2A task, detection and intelligent control agents play a role in supporting one or more of these behaviors. These in turn can be seen as roles performed by the commander, or as previously stated, they reflect an instance of the commander over some time frame. The commander switches between these roles to perform specific tasks as required. The application hardware acquires the necessary information and facilitates the action required for the agents to perform their role.

As defined previously, the information and computation resources are the "objects/components that generate, transform, store or distribute information required by object/components" of the entire system. It reflects the computational stack shown in Figure 34 along with the information support components. Additionally, a set of computation software resources beyond the operating system and resource access are included such as those typically associated with "middleware" or the aforementioned quality attributes, such as cyber security. The computational hardware resources include the types of computing or models of computation and mass memory as a specialized types of computation hardware support. Computing types could include any of the models of computation in Figure 24, but are currently focused on "general purpose processors" and "signal processors" as the main processing components of concern since direct control of sensors/actuators are distributed through the hierarchy. Other

computational types could be added as technology advances, particularly agent or autonomy based computing technology.

d. Mobility

The objects/components that generate, distribute and store the primary motive force or energy of the system; that support or are associated with movement; and that direct and control the movement relative to the external environment. Components include engines, transmission, headlights, suspensions, brakes, alternators and driving agents. Following the common agent pattern, the driving agents include a driving task agent (DTA), multiple driving detection agents (DDAs) and a single driving intelligent control agent (DICA). The DTA performs the role of the driver and the DDAs and DICA account for the required driver skills. The DTA interacts via commands/percepts with the mission agent and in turn interacts with commands/percepts with the DDAs and the DICA. They in turn interact with the mobility direct control systems which are allocated throughout the 3rd level under mobility and that interact with the physical-mechanical and power/energy components.

e. Lethality

Lethality is the objects/components that acquire targets and can generate and direct lethal and non-lethal effects against those targets. Component examples include direct fire cannon, fire control, munitions handling, mortar and lethality agents. The lethality agents follow the common agent pattern.

f. Tactical Command, Control and Communications (TC3)

TC3 is the objects/components that receive and process mission orders, report tactical situations and events to appropriate unit command levels, and that manage or support tactical information flow. Examples include FM radios, satellite radios, embedded tactical network systems and protocols, and agents or application software. Here again, following the agent pattern there is a TC3 task agent (TC3TA), multiple detection agents (TC3DAs) and an intelligent control agent (TC3ICA). Most activity surrounding planning for mission command (DA 72012) occurs via coordination between

unit commanders apart from the MIGVS system level. The execution and the detailed planning requires participation and interactions with many of the system resources, for both these reasons, the mission agent is included as part of SC3 and as opposed to TC3. The mission agent interacts via indirect logic as required to process the unit plan, conduct detailed planning, and execute the mission. The TC3 agents handle the information flow between the system and friendly units as required to support the mission agent. Unlike other agents, the TC3 agents only interact with friendly elements in the external environment and do not create effects in the external environment otherwise.

To the extent the system also includes roles that go beyond command and control of the system and to the extent that the resources of TC3 are the primary means of executing that command and control, the TC3 agents would include those additional "mission" agents. This would apply to upper levels of unit command such as battalion and company levels in maneuver units. The small unit level, such as platoons or sections, would have to be examined to determine the proper placement suitable for concept design analysis. For example, for a platoon leader managing hasty defense for his platoon resource responsibilities, may still rely on direct line of sight acquisition or require lower latency communication than that afforded from the tactical network.

g. Tactical Intelligence, Surveillance, Reconnaissance, & Target Acquisition/Electronic Warfare (TISR/EW)

TISRTA/EW is the objects/components that enhance situational understanding, evaluate threats, conduct surveillance and reconnaissance, and counter an external system's or threat's ability to conduct C3 and ISR activities. Examples include long range acquisition sensors, combat identification systems, and signal jammers. TISRTA/EW agents follow the same agent pattern, except there is no actuation that changes the state of the external environment. The agents correspond to roles and skills that are typically ascribed to the MIGVS commander.

h. Mission and Special Equipment (MSE)

MSE is different than the other second level objects in the MIGVS domain logical structural hierarchy that cover the common and/or combat logic for the domain. It is a

convenient collection of many miscellaneous objects that cover the remaining and more specialized logic possible in the MIGVS domain. The objects in the third level in general do not interact with each other and can be mutually exclusive, even to the fourth level of this hierarchy. An example is force projection, which is a mission area or type of mission that will bring with it a different set of objects/components. Two mission types are bridge launching and route clearance which typically would not be configured as a single system, rather it is integral to the unique identity of a single system Sustainment can include security, medical and transportation missions among others. Chemical, Biological, Radiological and Nuclear (CBRN) can be specialized objects/components on multiple systems and missions, and can be a set of objects/components that perform a specific CBRN mission. When instantiated for a given system, MSE objects/components at the third and/or fourth levels will be elevated to the second level of the system hierarchical logical structure and each may have its own unique agent structure.

For a MIGVS, the agent pattern can be related to types of roles or human positions, which in turn correspond to types of human or intelligent tactical tasks (DA 2012). In general, the mission agent corresponds to the management of crew collective tactical tasks, the task agents correspond to individual crew member tactical tasks, and detection and intelligent control agents correspond to skills required by individual crew members to perform all their tactical tasks. As all the agents are defined or the pattern detailed, the agents effectively correspond to all the roles performed by all the crew members. The sensors can be related to a given crew member's ability in the exercise of a role or task, to acquire the information needed to assess a situation and take appropriate action. The logic required by roles can now be modeled with a set of objects/components that is neutral relative to a specific physical solution, human or technological. Historical role types such as driving, gunning and commanding can be assessed in terms of autonomy, a mix of autonomy and human solutions, or a full complement of human solutions aided by information technology.

These agents are the superset of domain ALOs that were previously mentioned and they correspond to human roles and sub-roles. These roles must be abstracted from their "human implementation" while still preserving some understanding or model of the

behavior. This requires operational analysis on a comparative system. However, the roles performed are similar or common across many systems. Combining the agent pattern with the MIGVS logical structure yields the agents and roles that can be utilized to model behavior. An example select group of ALOs are shown in Table 4 with their corresponding domain logical performer group and human roles or positions. A significant level of granularity of human behavior is thus delineated.

Table 4. MIGVS Agents, Logic and Human Positions

Agent	Logical Group	Human Position
Mission	SC2A	Commander
SC3 Task	SC2A	Commander
TC3 Task	TC3	Commander
TISR/EW Task	TISR/EW	Commander/Specialist
Driver Task	Mobility	Driver
Lethality Task	Lethality	Gunner
Bridge Launch Task	MSE-Force Projection	Specialist
Material Handling Task	MSE-Sustain-Transportation	Multiple
CBRN Task	MSE-CBRN	Multiple/Specialist
SC3 Intelligent Control	SC2A	Commander
TC3 Intelligent Control	TC3	Commander
TISR/EW Intelligent Control	TISR/EW	Commander/Specialist
Driver Intelligent Control	Mobility	Driver
Lethality Intelligent Control	Lethality	Gunner
Bridge Launch Intelligent Control	MSE-Force Projection	Specialist
Material Handling Intelligent Control	MSE-Sustain-Transportation	Multiple
CBRN Intelligent Control	MSE-CBRN	Multiple/Specialist
SC3 Detection*	SC2A	Commander
TC3 Detection*	TC2A	Commander
TISR/EW Detection*	TISR/EW	Commander/Specialist
Driver Detection*	Mobility	Driver
Lethality Detection*	Lethality	Gunner
Bridge Launching Detection*	MSE-Force Projection	Specialist
Material Handling Detection*	MSE-Sustain-Transportation	Multiple
Chemical-Biological Detection	MSE-CBRN	Multiple/Specialist
Nuclear Event Detection	MSE-CBRN	Multiple/Specialist

The asterisks (*) for the detection agents in Table 4 indicate that the possibility of multiples. Detection agents need to be defined relative to the skills that they embody, what needs to be detected, and the uniqueness of the associated sensor(s). For example, a many driving tasks require the need to stay within lanes appropriately. This implies the

need to detect lane or edge markings and their types or implications. Mobility tasks also require reacting appropriately to the dynamic environment, such as the behavior of other vehicles on the road. This implies the need to detect other vehicles and interpret their behavior. The sensor implications of these distinct detection tasks suggest the possibility of different orientation and focus. Though these detection and sensor capabilities are integrated into a single human solution, a human solution can be considered to be a relatively expensive solution. It is also possible that distinct sensor solutions and positions can yield better than human performance standard. The identification of specific domain detection agents will have evolve over time with application of the domain hierarchy, project and operational analysis, and autonomy and sensor technology maturation.

There are also numerous mission types each of which could have a task agent, an intelligent control agent, and multiple detection agents. In the case of I/RSTA/EW and CBRN, these can be specialized capability of a system or the primary mission of the system, such as a scout vehicle or CBRN reconnaissance vehicle, respectively. These specialized vehicle system missions are quite numerous, so just a few examples are listed in Table 3, a bridge launcher system and a transportation tactical system that includes material handling for managing transported supplies on and off the vehicle. These systems will have much of the logic that are in common for all combat and tactical MIGVS systems, but will have the additional agents, underlying direct control objects, and sensors and actuators required to perform the specialized tasks. Finally, there are potentially more agents lower in the hierarchy, such as for Survivability/Hit Avoidance/Active Protection and Lethality/Munition Handling, as well as the aforementioned unit command agents whose role might be assigned to a given MIGVS.

The agents along with the other logical objects in the MIGVS structural hierarchy form the “pick list” to instantiate any given MIGVS system. The agent objects enable the system to be instantiated independent of whether the intelligent behavior is to be realized by human operators or technology. Each of these ALOs can be thought to correspond to a physical instance of the human position or can be assigned to a hardware and software physical solution that represent a set of leaf level objects. They can reflect the “as is”

behavior of a comparative system or to postulate new behavior with new tactics. All the objects reflect a minimal level of behavior or basic purpose and properties that can be used to assess candidate specific component or technological solutions.

2. System Context Reference Model

In much the same way the system performer logic can be assembled as a superset of domain objects, the external context can be similarly objectified for the MIGVS domain. The superset of MIGVS domain objects, referred to as a System Context Reference Model (SCRM), are shown in Figure 53 for the first three levels of logic. These objects can also have attributes that reflect the state of the external context when defined at a moment in time, or that at least reflects the relevant state. As with the system domain structural model, this hierarchy can be decomposed until a specific physical thing or entity is defined. Conversely, depending the level of modeling abstraction, relatively high levels of entity abstraction could be considered leaf level with lower entities modeled as properties. As complex as defining the relevant world might seem, the U.S. Army has done this in various forms, to include doctrine (DA 2008) and command and control (JC3IEDM 2007), which were leveraged in the generation of the model in Figure 37.

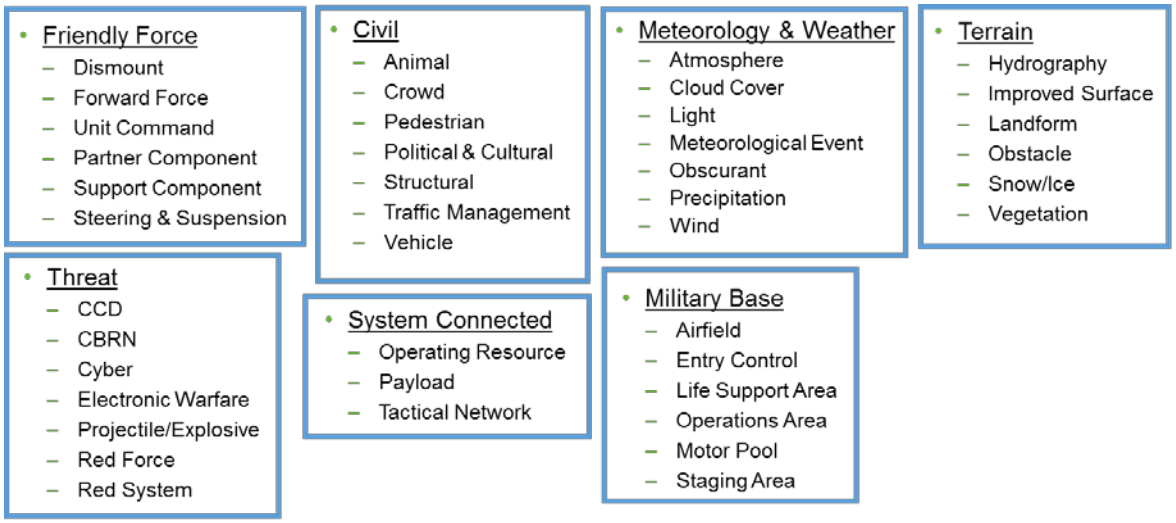

Figure 53. MIGVS Domain SCRM

The SCRM like the domain system performer structural model can be used as a "pick list" of external context entities for system interactions. Together they represent an overarching world model for the system that can also be used to model both a goal state and belief state. These context objects change state when they produce events and when they absorb effects. The system objects produce effects to context events as needed. The second level domain context objects are defined as follows:

a. System Connected

System Connected Context is defined as context entities have overlap with the system boundary at some point in their life cycle. As indicated before, operating resource is used or consumed during system operation and includes material, energy, and information. The entity abstraction physical can be used for material and energy so as not to bias technology selection. Information is externally generated information that is used in some form by the system. Examples include mission orders, maps and control symbols (DA 2004). Payload includes operators, passengers, and material that is not an operating resource. The Tactical Network exists distinct from the system though the system has an interface component or embedded node.

b. Friendly Forces

Friendly forces are the set of mobile physical objects or "systems" that the MIGVS interacts in order to achieve its mission goals. These physical objects are further defined according to the role they perform relative to the MIGVS of interest: subordinate, command, lateral/team, support and dismounts. There is typically an interdependence of goals between these systems that requires coordinated planning and synchronization. Each of these objects can be service, joint, coalition or civil systems as indicated by their attribution.

c. Threat

Threats are the set of physical objects that can cause a MIGVS harm through its interaction during a military operation or as a consequence of being a military system. They can range from something as physically concrete as a projectile, to something more

nebulous such as a cyber security object, which can be mean the source of an on-going attempt at penetration, or an active cyber threat once penetrated.

d. Civil

Civil is the set of objects related to non-combatant human and societal elements that a MIGVS may interact with as an indirect result or consequence of a military operation, particularly in an urban environment. These objects can be virtual such as political areas, symbolic such as some traffic management components, and physical. Urban environments have a greater variety and density of objects than non-urban environments.

e. Terrain

Terrain is the set of objects that the MIGVS interacts with as a direct consequence of its mobility and has indirect impact on other system logic. These objects can overlap, but can be considered as distinct interactions such as vegetation on a landform, or have a dominant interaction such as snow/ice on a road.

f. Meteorology and Weather

Meteorology/weather is the set of natural and man-made objects that the MIGVS interacts with in the environment above the terrain, primarily related to atmosphere or air. These objects interact with structure in terms of protecting internal components, manipulators or actuators relative to effect creators, such as lethality, and any sensor as regards visibility.

g. Facilities and Infrastructure

Facilities/Infrastructure is the set of objects that the MIGVS interacts with the course of its life cycle that can be deemed “friendly” beyond friendly forces. They can overlap with similar civil objects, but are tracked differently since there attributes are more controllable relative to a MIGVS solution. These objects together can constitute an army post or military installation or some facsimile thereof.

3. MIGVS Concept Design

Concept design is defined here as a system design abstraction that identifies major configuration items, system and configuration item allocated behavior, and the underlying implementing technology. Concept design can represent a baseline configuration from which detailed physical and behavior design proceeds. Concept design can also be used as a preferred system alternative to support a competitive acquisition in terms of program feasibility analysis, cost estimating, etc. The concept design phase generally includes one or more system conceptualizations that are analyzed via trade studies supported by a variety of analytical and simulation based techniques. Models in 3D CAD form a critical part of this conceptualization and support physical integration feasibility, mass property determination, and low level physical behaviors.

a. Initial Concept Design

The embodiment of cyber concepts: crew size, computational architecture, sensors, software, etc., not only impacts the mass properties of the system, but impacts the overall performance and operational effectiveness of the system. It has not only its own trade space to consider, but has an interdependency of trade space with the physical concepts and component solutions. The initial concept design of the cyber behavior must enable the generation of alternative cyber concepts and integrate when needed with the physical concepts. The analysis of physical and cyber are supported by different methods and techniques but must be periodically and finally integrated into a whole system concept. The initial system concept must enable, but not constrain the subsequent embodiment of the system and be used as a point of departure of cyber capability if required by the trade space analysis.

The role of initial concept design is to capture the operational behavior and any required system behavior at a concept level of component abstraction. This concept level of component abstraction forms the initial concept EBOM with attribute types for the system that can be used to construct the initial physical EBOM of a CAD model and the initial “cyber” EBOM. Each EBOM can be expanded and/or adjusted through the generation of physical and cyber concept alternatives and specific attribute values as a

result of analysis and trade space exploration. The cyber concepts have physical constraints in the form of volume available, distribution of components, etc. The physical concepts have cyber constraints in the sense of whether the final integrated physical concept meet the needed operational behavior. Both the cyber and physical concept types need their respective separate identities to facilitate different types of analytical methods and they need to be integrated to facilitate a coherent system concept and interdependent trade space. Note that the initial EBOM does not do much to inform the physical concepts and the latter could be conducted more concurrently as the necessary recursion and integration can occur.

b. System 4+1 Model

The final concept design can be illustrated in a System 4+1 model as shown in Figure 54. This is similar to one of the many UML 4+1 versions as shown in Figure 3 with some notable exceptions. The initial concept design is the initial logical view. As the concept design is formulated the initial logical view leaf level can be appended with specific implementations or technology and cyber attributes assigned values. The Process View represents the assembly and technological constrained logical view which will impact its ability to meet key operational and system behavior. As cyber components are physical realized, resource limitations and distribution will constrain the cyber performance. The Physical View is the more standard EBOM view would be ideally augmented with software assembly and components. The Deployment View represents the physical deployment and distribution of the EBOM as typically found in the CAD model. This Deployment View would ideally also include the software deployment to computing resources.

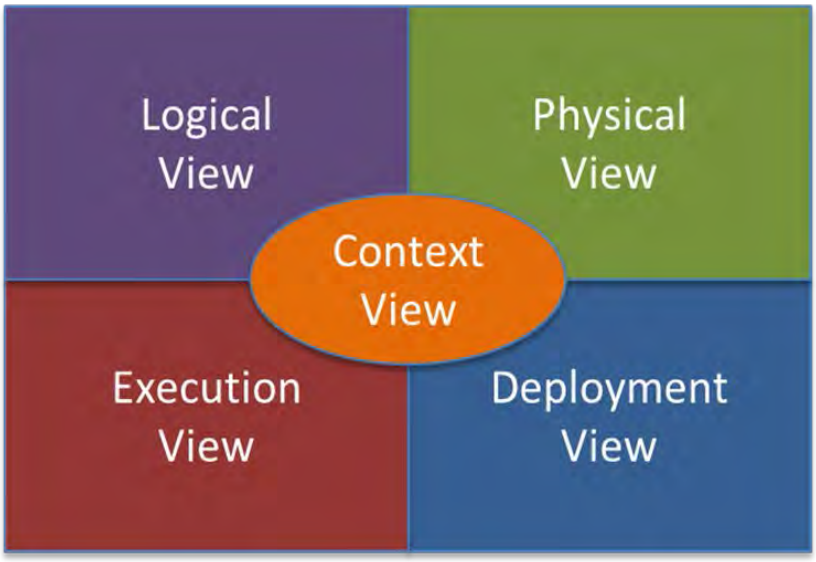

Figure 54. System 4+1 Model

Note that in the four views just described should have the same leaf level set of components. The difference is in the rollup of the components and how that roll up is used. For instance, the rollup of the Physical View is supports physical assembly and installation where the rollup of those same components in the Logical or Process View facilitates operational and system behavior analysis. The leaf level component represents the elementary component appropriate to the design stage abstraction. In many natural system hierarchies (Simon 1962), what is considered elementary is somewhat arbitrary, much like the atom was once considered the elementary component of matter. It can vary based on understanding, interest and other factors. This notion can extend to distinguish between the elementary objects that need to be defined for concept design versus engineering design. Concept design is the focus of this research and its elementary level is the point where specific technological or physical solution is introduced into the hierarchy. The elementary level for engineering design reflects part level detail final design and a configuration managed EBOM. If desired, the logical structure design could be elaborated to this level of detail, but this level of detail would not be required for concept design. Concept design does require selection or object instantiation with a component via a certain types of technology, to include some semantic information.

The "+1" view is the Context View. As mentioned previously, the equivalent of "use cases" is now embedded into the initial Logical View as goals. The context is critical to not only defining goals, but also serves the more traditional engineering purpose of bounding the system and identifying the system's external interactions. An objectified context produces events and absorbs effects as its attributes change. Most operational goals reflect some desired effect or state of the context. The context's state attributes can also be linked to METT TC variables (ADRP 3–0, 2012) that correspond to military planning and scenarios. Ideally, the context used in any analysis or simulation should track to the context that drives the system behavior. This warrants a distinct view of the context so as to facilitate translation if not have the same structure.

4. Palletized Loading System (PLS) and Convoy

The PLS A1 M1075A1 (Oshkosh Defense 2017) is a wheeled vehicle system with a built in Load Handling System designed to enable "supply and equipment distribution." It is essentially a truck with a built-in crane that can also be configured with a variety of mission equipment. It can load and unload flat racks that have standard Army pallets. Distribution of supply is almost always conducted as a convoy mission. The M1075A1 will be used as a design reference system and convoy as a design reference mission for this case study. The case study will amplify system and mission information based on identified references and expert opinion and translate them for capture in a SysML model.

a. Design Reference System

A *design reference system* is defined as an existing system whose crew and system capability are abstracted into a solution neutral component based logical or initial concept design. The PLS A1 vehicle system has similar components to any truck system in terms of mobility, structure and survivability, can accept most of the Army's standard mission equipment in terms of TC3, I/RSTA/EW and lethality, and is typically operated by a crew of two. The LHS system gives the PLS its unique syntactic or structural logic identity. The structural logic and behavior vehicle, crew, and mission equipment are reversed engineered and abstracted into a set of solution neutral components. The crew

and crew behavior will be represented by an abstract set of ALOs and sensors that operate and control the other systems as appropriate. This logic can be utilized to analyze a particular cyber-related upgrade concept to the design reference system or used a baseline reference to support any replacement concept evaluation.

b. Design Reference Mission

A particular convoy mission, like any Army mission, is defined using a mission operations order that defines the purpose, expected threat, route plan, and other METT TC variables. The convoy will also include a manifest that describes the supply cargo that each PLS is to carry. The convoy can be configured into multiple March Units, will have a Convoy Commander, and may be augmented with gun trucks depending on the threat and other support vehicles. The focus here will be on the PLS that acts as follower vehicle within the convoy versus a leader-type system. Most of the PLS vehicles will function as follower systems in any convoy. The mission will begin with an operations order received by each follower system in the motor pool. The follower system will be required to obtain its cargo, follow appropriately within a convoy, and then deliver the cargo to some designated destination.

Deliver or effect supply is the mission effect of the convoy overall and the follower PLS specifically. The convoy mission is synchronized and coordinated both internally and with appropriate external elements as part of the a priori mission plan and through its execution. The synchronization and coordination is accomplished through standard crew and military discipline behaviors such maintain tactical situation awareness, maintain effective communications, follow the route plan, follow the lead vehicle at distance and speed, etc. Additionally, certain “rainy day” behaviors are expected to be handled when encountered, such a avoiding an obstacle. These behaviors will be defined as explicit goals with desired outcomes in the state of the context and the system. Sources for convoy behaviors (ATP 4–01.45 2014) and crew behaviors (TC 21–305-10 1994) will be from public information or highly generalized to basic behavior. The objective is to demonstrate the concept and not be an expert treatise on convoy behaviors.

B. AGENT- AND OBJECT-BASED PLS CONCEPT MODEL

A general approach can be defined for developing a system's initial logical concept design. This approach can be defined in terms of processes associated with model elements or artifacts. Though an order in general can be defined, many of the processes can begin independently and execute concurrently. Many of the associated artifacts overlap or are interdependent in terms of their elaboration. Also, there is both a "real physical" representation and virtual or data representation of certain classes or objects within the same model. The overall approach requires synchronized model elaboration among multiple artifacts or diagrams as they are initiated, linked, elaborated and finalized. There are artifact integration dependencies horizontally between them and hierarchically between the same artifacts for different agents. Since all model artifacts are "objectified" including missions, trajectories and goals, new objects can be introduced and added to the existing objects as required without undoing the previous work. The general approach is address the key "sunny day" mission and military discipline trajectories first and afterward address exception handling and any detailed behavior elaboration as needed. The proof of concept for this research is based on the key sunny day mission.

The actual fidelity of the agent and object based concept model to support an acquisition program depends on the model's purpose. Structural fidelity can be expressed in terms of breadth and depth of the objects as well as the attributes that define the state space of the objects and/or timing related performance. Behavior fidelity is expressed in terms of the mission and desired trajectory decomposition along with the interactions of the objects. This research has previously identify three purposes and each purpose requires differing degrees of fidelity:

(1) Purpose #1: Inform and augment the initial need or operational capability specification. Design reference missions are defined in conjunction with the user. Design reference missions should include critical behaviors or desired trajectories and associated state and time attributes. State attribute details and non-ALO object attributes are not needed for initial operational specification

(2) Purpose #2: Support pre-award concept design feasibility. Identify key attributes of the behavior and the system components required to support a trade study. The selected technology should be identified and linked to the technology neutral component objects. The component object attributes should be defined where they match the trade study criteria used to evaluate the technology.

(3) Purpose #3: Define a logical design to be part of overall system technical baseline. All component objects defined and fully attributed in the technology neutral model or the logical concept design model. The logical concept design model should be extended with the selected leaf level technology selection. The leaf level components or parts will be defined in common with a CAD model engineering bill of materials (EBOM). The logical design will form a decomposition BOM (DBOM) view into the system's components and parts.

Assigned Mission Model. The pattern for an assigned mission state model was shown in Figure 41. The model defines the mission for a reference system (e.g., PLS), in terms of key tasks and their desired trajectories. It orders the trajectories and any sub-trajectories according to their execution across mission phases and links goals to the desired state space. As the desired trajectories are decomposed, the goals ae typed as achieve or maintain goals and goal specification state measures are defined. The goal state measures reference properties in the selected state space. The state space consists of system performer and context objects and their attributes and elaborated as needed to define goal measures. The initial design reference system mission model is allocated to the Mission Agent (MA) and then apportioned out to the supporting tasks agents. The desired trajectories are successively elaborated and missions allocated to lower level agents until the control and sensor desired trajectories are defined.

Define Logical Object Hierarchy. The logical object hierarchy consists of system performer objects and context objects. System performer objects are selected based on the design reference system and the domain logical hierarchy of Figure 52. If a suitable logical object cannot be found, a new logical object can be created and added to the domain logical hierarchy. The performer objects should be selected based on the mission and desired trajectories identified previously. These object can be alternately expanded as part of this overall hierarchy or expanded selectively as it is included in the state space of one or more desired trajectories. A given performer objects attribution is likely to be

expanded as it is needed to support a given trajectory. There should be one mission agent, a task agent for each top level system performer object, one intelligent control agent per task agent, and as many detection agents as required to realize the desired trajectories. Finally, the controls and sensors should culminate the selection of performer objects. These too will be defined in conjunction with mission and desired trajectory elaboration. The "controls" and "sensors" are actually combinations of control system, power amplifier, and actuator; and sensor controls and data conversion, respectively.

In similar fashion to system performer objects, the context objects should be selected from domain context hierarchy of Figure 53 and from a review of the operational source material. A key objective is understand how the operator(s) of the design reference system views the external world, particularly as regards METT TC variables. The object and attributes are also elaborated in conjunction with their use in the state space of the mission and desired trajectory elaboration. The set of all performer objects fully elaborated and attributed together and the set of all context objects fully elaborated and attributed constitute the system concept data model. These objects pulled from a common reference and elaborated as state space for the mission and the various desired trajectories, will result in that common reference being a fully elaborated concept data model.

Horizontal Interactions. The static behavior of the system is captured as a set of signal flows between performer objects of the system. As explained previously, these signal flows are of two types: horizontal or indirect and vertical or direct. The horizontal signal flow between performer objects are the interactions that drive state change within the objects and are captured using SysML IBDs. ALOs horizontally interact between each other and with controls and sensors using commands and percepts. These commands and percepts type the signals that flow between proxy ports that are types by ALO interface blocks. The mission agent horizontally interacts with the unit commander and the system task agents, the task agents with intelligent control and detection agents, etc. Each performer object set has a set of IBDs that define the interactions. These performer object IBDs are connected via an IBD with the mission agent. Together the entire set of

IBDs constitute the system behavior threads that encompass events and effects as defined by desired trajectories.

Agent Internal Composition Models. Each performer class has one task agent, one intelligent control agent, and one or more detection agents. Additionally, the SC2A performer class also has the mission agent. Each one of these ALOs are modeled as SysML BDDs to include a behavior and a world model. As shown in Figure 43, the world model includes knowledge and world state model which in turn is composed of assigned world state model (AWSM) and derived world state model (DWSM). Knowledge includes standard attribution used in the computation of goal state constraints. The AWSM contains the ALO's assigned mission and trajectories and the DWSM includes the assigned mission and trajectories of its subordinate agents. Note that with the DWSM allocation of trajectories that the decomposition of goals with allocation to agents is effect. Also, identified are the commands and percepts coming from and into the agent. These are the same commands and percepts that were types as signals in the IBDs.

Agent Behavior. At the top level, the agent's behavior is simply to manage its assignment, derive subordinate agent's assignments, and manage their activities. It's is essentially managing data in the form of world state models and the reception and issuance of commands and percepts as indicated in Figure 49. The agent must be able to evaluate the belief state and goal constraints and know what action to take as a result. From a concept design standpoint however, it is assumed if the agent has access to the necessary data, behavior can be designed to take the appropriate actions. The key to concept design is defining the necessary data in and out of each ALO and insuring there are sensors and controls appropriate to that data. The agent's AWSM as assigned to it from the upper agent composition model is a parameter node that this agent's behavior must access and update. This agent's DWSM defines its lower agent AWSMs in its internal composition model which also types parameter nodes that the agent must access and update. Finally, the commands and percepts from the various IBDs and agent composition BDDs are also parameter nodes coming into and out the agent behavior.

Vertical Interactions. As indicated in Figure 34, the indirect logical interactions must go through several direct transformations up, down and across a computational

stack and signal distribution from source to destination. The SC2A performer class defines these classes and objects for the system overall where they can then be directly connected to the other performer classes. Each performer class will add computation and signal classes to it performer-context pair and then elaborate direct vertical interactions as required. These vertical interactions are still abstract and logical as are the SC2A classes. The idea is to understand the logic that will need to be examined as specific computation and signal architectures are defined and assembled. This will represent a constraint on the horizontal thread performance that can be examined in a trade study. The SC2A contains the entire system's computation and signal objects and their interactions. It can also receive commands from the mission agent and have goals. Goals can include providing computation and signal performance, availability, status, etc. A similar approach can be defined for power distribution to the system loads, but will not be explored in this research.

Integration and Iteration. As was mentioned, though a general procedure can be defined, there is much interdependency between the associated modeling artifacts. A key set of logical structural dependencies is shown in Figure 55. The overall logical structure is composed of the system and context. The system is composed of a set of performer objects of which an agent is a type. An agent is composed of agent behavior and a world state model. The world state model is composed of an assigned mission that has one or more desired trajectories, each of which has one end state goal. The goal has state measures that reference a state space which in turn references the system and context objects of the logical structure. The agent behavior operates on parameter nodes: AWSM which is part of the agent world state model that reflects its assigned mission, a command that goes to a subordinate agent via a signal and that contains its allocated goal state values, and a percept that goes to a superior agent via signal that contains current state values to that same goal state space.

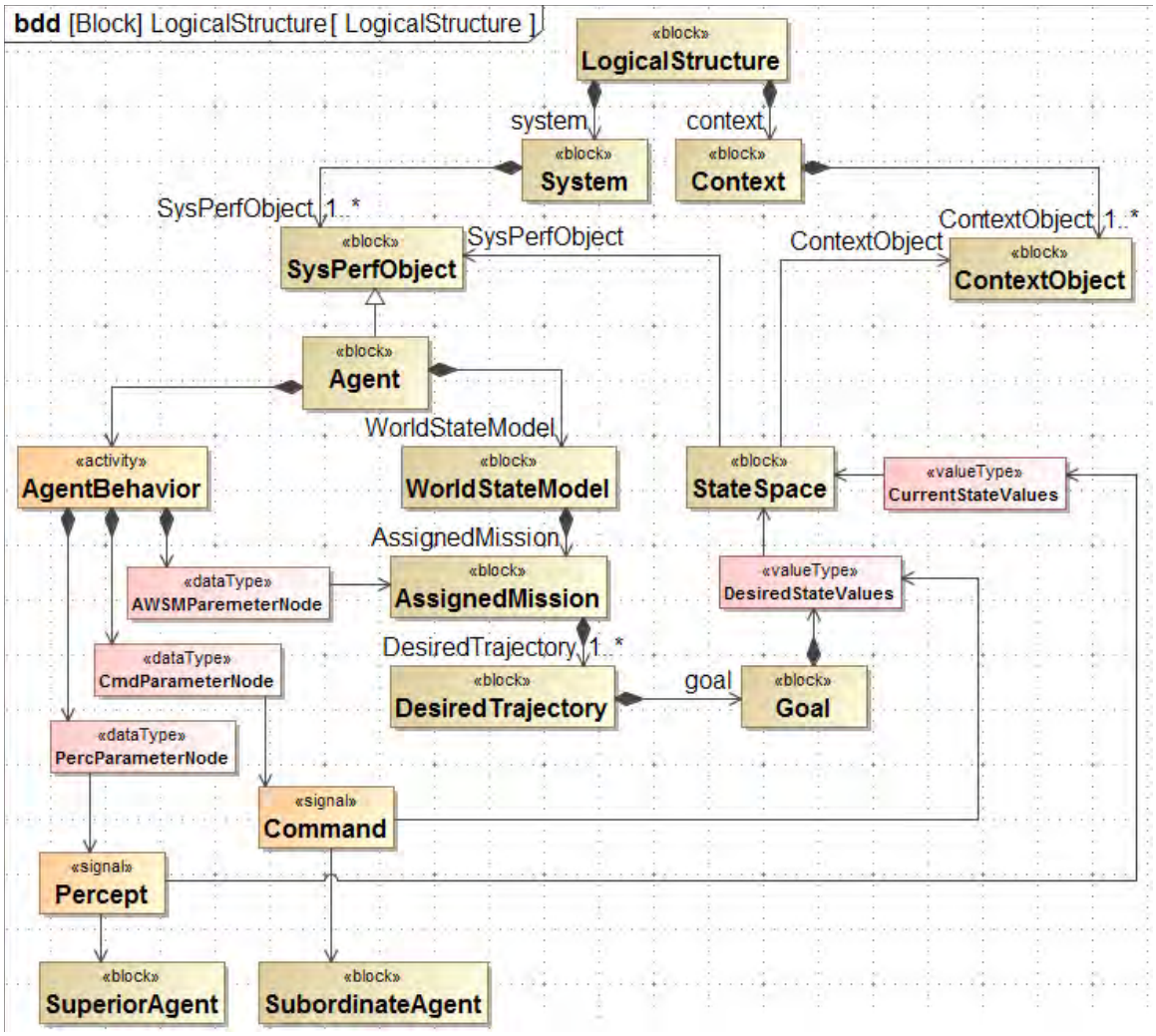

Figure 55. Agent Model Structural Relationships

There are other relationships between model elements that are not shown. These include: the DWSM behavior parameter nodes and composition part of the WSM that reflect the subordinate agent assigned world models, world model knowledge that's part of agent composition, percepts coming from subordinate agents, commands coming from superior agents, ports on IBDs that are typed by agent interface blocks, and the commands and percepts that type signal flow on IBDs. All these relationships between model elements need to stay linked as they are elaborated. It is easier, if not necessary, to go back and forth between model elements as the decomposition of each progresses

rather than try to decompose to the lowest level of one model element. Note that in the iteration, much logical decomposition takes place.

What follows is the development of these respective model elements relative to key PLS performer objects. All model elements: assigned mission, logical object hierarchy, horizontal interactions, agent composition and agent behavior, are first defined relative to the system and the mission agent. Then more elaborated versions of the same model elements are defined for material handling. Finally, the other performer objects are addressed. Each performer object elaboration is augment with an appendix where the full description of the model elements can be seen.

1. PLS and Mission Agent

The mission agent (MA) by definition manages the mission for the system and sits atop the logical behavior hierarchy. As such, the mission agent receives the PLS assigned mission and is the top level logical interface between the system and unit and is responsible for overall mission effectiveness and performance.

a. PLS Assigned Mission

The assigned mission to the PLS is convoy follower. The PLS assigned mission orders and phases the tasks and desired trajectories required to perform the convoy mission. The major tasks are supply effect, tactical situation awareness, command synchronization, various forms of maneuver, and provide autonomics. Each of these tasks include a set of desired trajectories or even recursive desired trajectory hierarchies. Tasks and trajectories include a portion of the assigned mission state space. The assigned mission includes a state space that bounds the system interest in terms of performer and context logical objects. The desired trajectories and state space are further decomposed and detailed within each performer object. All the relevant information from the mission order is included in the assigned mission and may include some tasks that would typically be part of institutional or unit training of human operators, such as military driving. The latter enables greater adjustments to mission rules and conditions if needed; that is, the system can take on more or less risk in the performance of a given desired trajectory depending on overall mission criticality.

The assigned mission has a variable that identifies the current phase: planning, preparation, execution and refit. Each task then initiates in some planned order via an order number that also identifies the phase or phases. In turn, each desired trajectory within the task and each sub-trajectory with the desired trajectory also initiate within some planned order. The tasks will typically initiate early in planning or preparation and last until refit and run concurrently. Desired trajectories will be shorter, but may occur several times over the mission plan and may execute in sequence or concurrently with other task desired trajectories. For a supply effect task example, it has three top desired trajectories, acquire supply which occurs during preparation and transport supply and deliver supply occur during execution, though the latter constitutes the end of the execution phase and the release from the convoy. The mission agent orchestrates this activity and all planned behavior should execute in the proper phase and order barring unplanned events. In this sense, the mission order as planned represents the sunny day behavior of the system.

Examples of training tasks and non-sunny day behaviors are mission exceptions (i.e., events that occur which cause a departure from the mission plan, at least temporarily). Response to these events are not planned in the mission order by definition, but are known to happen per doctrine. As shown in Figure 56, identified exceptions are leader error, obstacle avoidance, defensive position, and rally. Many more exceptions are possible. Though not part of the plan, the system must respond predictably to these events. In this sense, they can be identified as a task that gets invoked similar to an interrupt. The exceptions will not be elaborated with the rest of the model, they are only illustrative of how they would be handled by the system using this approach. Each specific exception would be identified and linked to an interrupt. Any aspect of the standard mission that competes with the exception will be at least temporarily suspended. Once the exception is addressed, the system will try to return to the original mission order and provide a report back to unit command indicating its status relative to achieving the mission order goals. Unit command can issue new instruction via a fragmentary order if need be, and a new plan would be initiated.

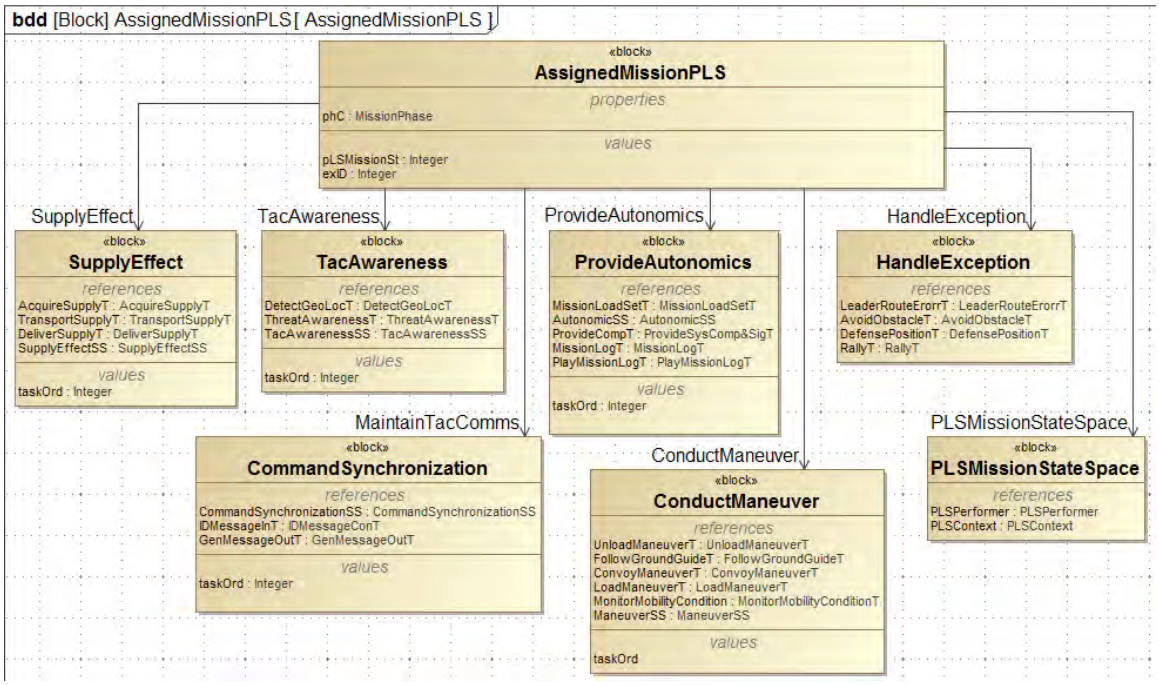

Figure 56. PLS Assigned Mission

b. Mission Agent Logical Object Hierarchy

The mission agent logical object hierarchy is shown in Figure 57. It bounds the relevant state space and when fully elaborated, also serves as the system concept data model. It consists of both performer logical objects and context logical objects. Since the mission agent orchestrates the overall mission and associated goals, it also must be cognizant of the entire mission state space, at least at some level of abstraction. Lower level tasks and desired trajectories pull from this model as required and may elaborate both entities and attributes. Per the modeling language and tool, they become part of an elaborated concept data model during the elaboration process.

Note that although the mission agent is cognizant of the entire state space, it only knows current state based on task agent percepts and goal state based on convoy task force command; that is, it only indirectly logically interacts with the task agents and convoy task force command and does not sense the environment directly. The mission goals are divided and allocate among the task agents for each top level performer: material handling task agent (MHTA), tactical C3 task agent (TC3TA), IRSTA task agent

(IRSTATA) and the system C2 and autonomics task agent (SC2TA). The top level context objects are system connected, friendly force, civil, terrain, meteorology and weather, and both a source and destination military based. The mission agent tracks the state of the context either from external tactical reports that are augmenting information from orders it receives, or from performer object sensors that are passed up through the agent hierarchy.

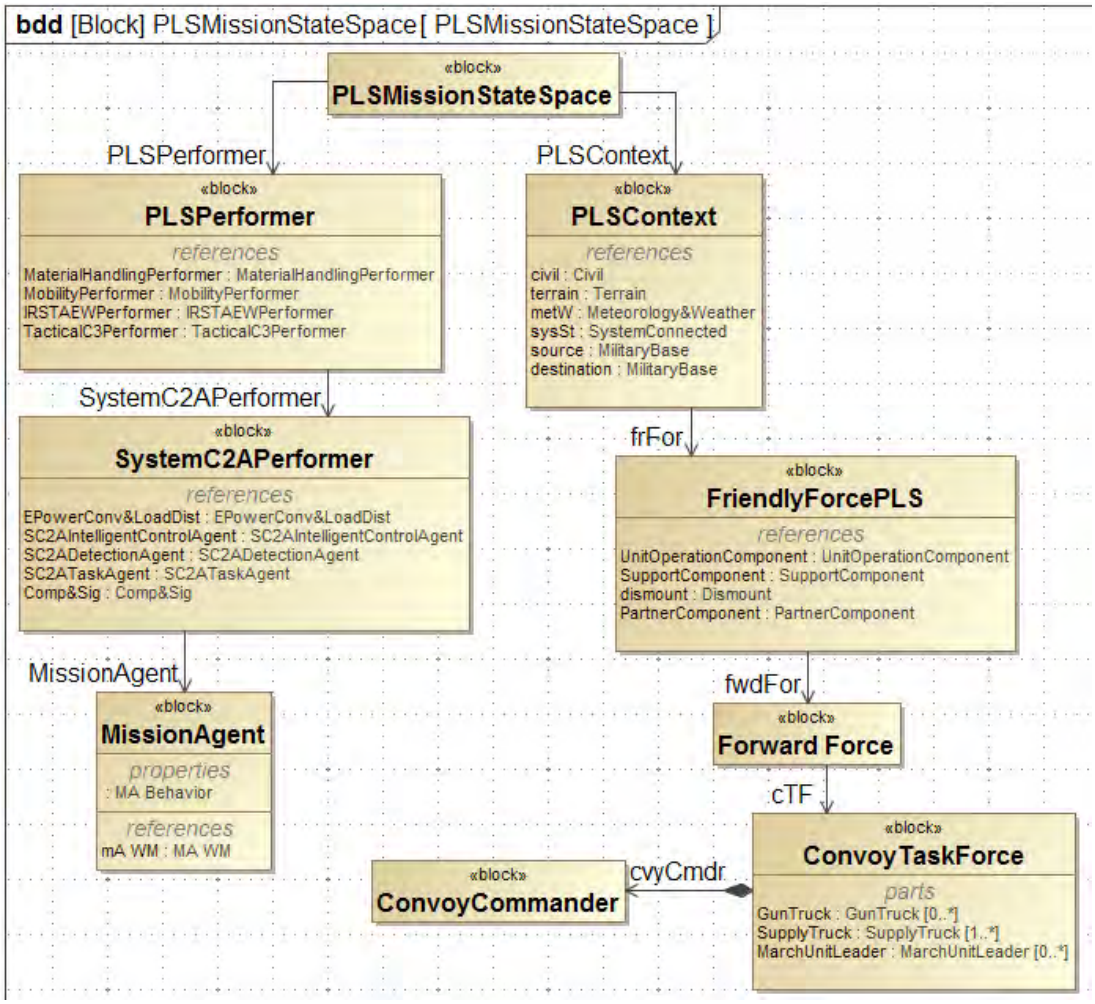

Figure 57. Mission Agent Logical Object Hierarchy

The system performer logic elaboration is shown in Figure 58. It represents a selection from the domain logical hierarchy shown in Figure 52. It identifies the full complement of performer types for each top level performer: sensors, controllers, and all

the three agent types: task agent, intelligence control agent and detection agents. Each performer logic is subsequently elaborated as part of that performer's overall assigned mission and state space elaboration (i.e., material handling only elaborates itself, not other performer objects).

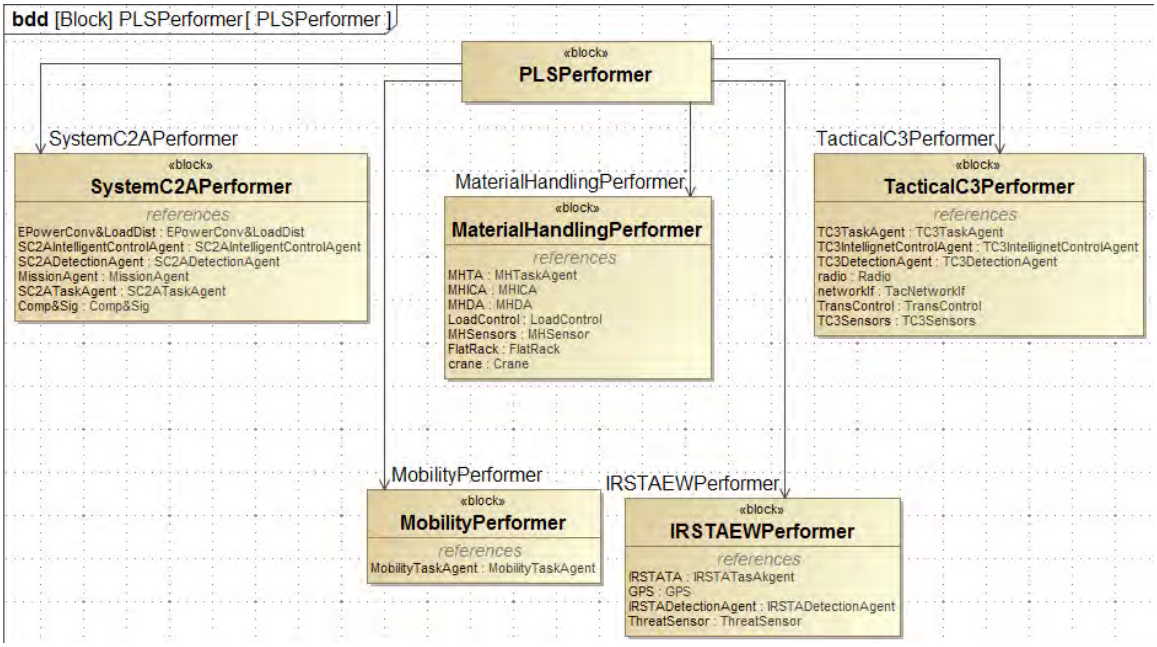

Figure 58. PLS Performer Logical Object Hierarchy

The PLS context elaboration is shown in Figure 59 as selected from the domain context logical hierarchy of Figure53. Each performer logic elaboration mentioned above, may elaborate or add any object in the context. The duplication is realized only in the individual performer object hierarchy, the state space of Figure 57 will only have the unique additions.

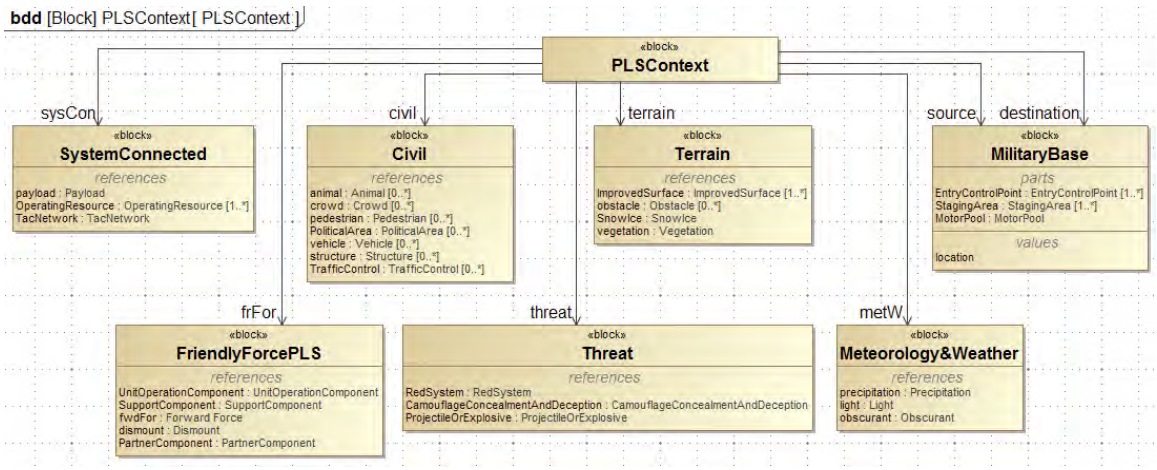

Figure 59. PLS Context Logical Hierarchy

c. Mission Agent Horizontal Interactions

The mission agent horizontal interactions are shown in Figure 60. These are shown as any other signal interaction in SysML on an Integrated Block Diagram (IBD). All interactions further explored will of this same type except for where direct interactions are specifically identified, and where controller and sensors interact with any plant objects or context objects, respectively. As mentioned previously, the mission agent horizontally interacts with each task agent and the convoy commander. The proxy port naming convention is based on what it connects to rather than where it is located. However, the ports are typed by interface blocks of the lower agent for the commands and percepts. For example, the command and percept "signals" between the mission agent and the MHTA are owned by the material handling interface blocks and referred to as MHTACmd and MHTAPerc. The signals or connectors between agents are typed with those commands and percepts and given the appropriate direction as shown in Figure 60.

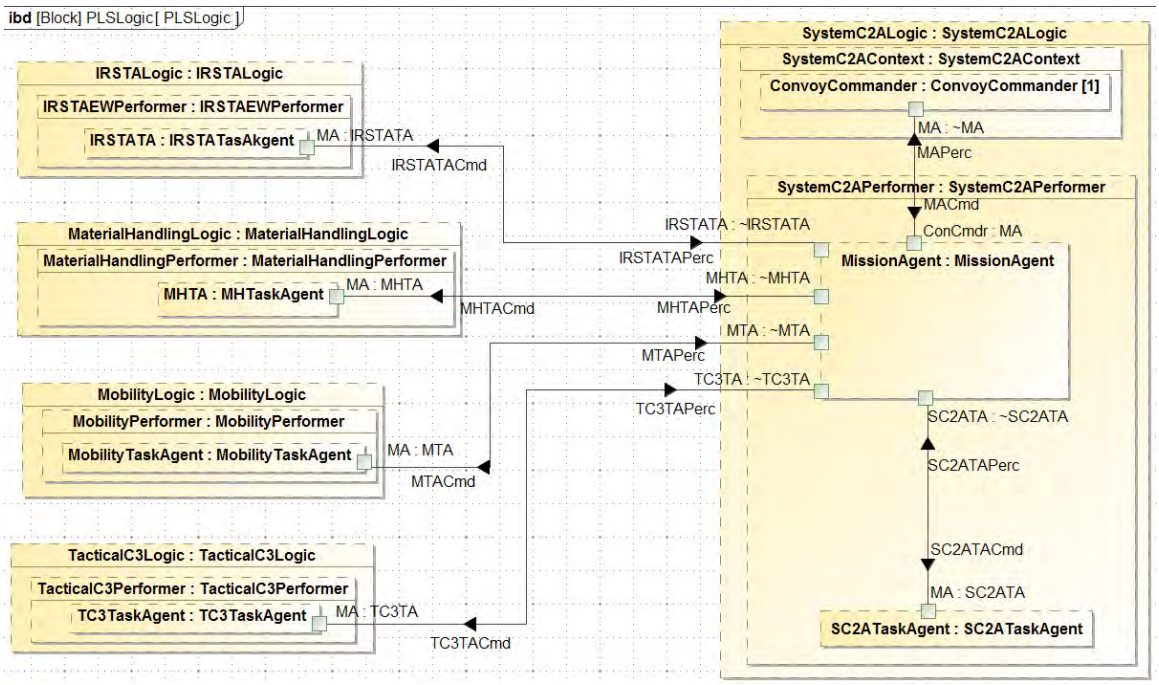

Figure 60. Mission Agent Horizontal Interactions

d. Mission Agent Internal Composition

The Mission Agent (MA) internal composition shown in Figure 61, like any ALO, follows the general agent composition pattern of Figure 43. Its assigned world state model (AWSM) is the assigned PLS mission of Figure 56. The MA derived world state model (DWSM) is composed of the various task agent AWSM models and they contain the mission tasks allocated to them. Each of those tasks are elaborated as needed. As shown in Figure 61 for example, the supply effect tasks has three top level desired trajectories: acquire, transport and deliver. The acquire supply trajectory in turn has 7 trajectories. These desired trajectories along with their respective goals and state space are elaborated as part of Material Handling, though in operation, the upper level trajectories are assigned as part of the mission order. The commands and percepts from the horizontal interactions of Figure 60, are linked to the appropriate AWSM model. These commands and percepts communicate goal state and current state of their respective AWSM model.

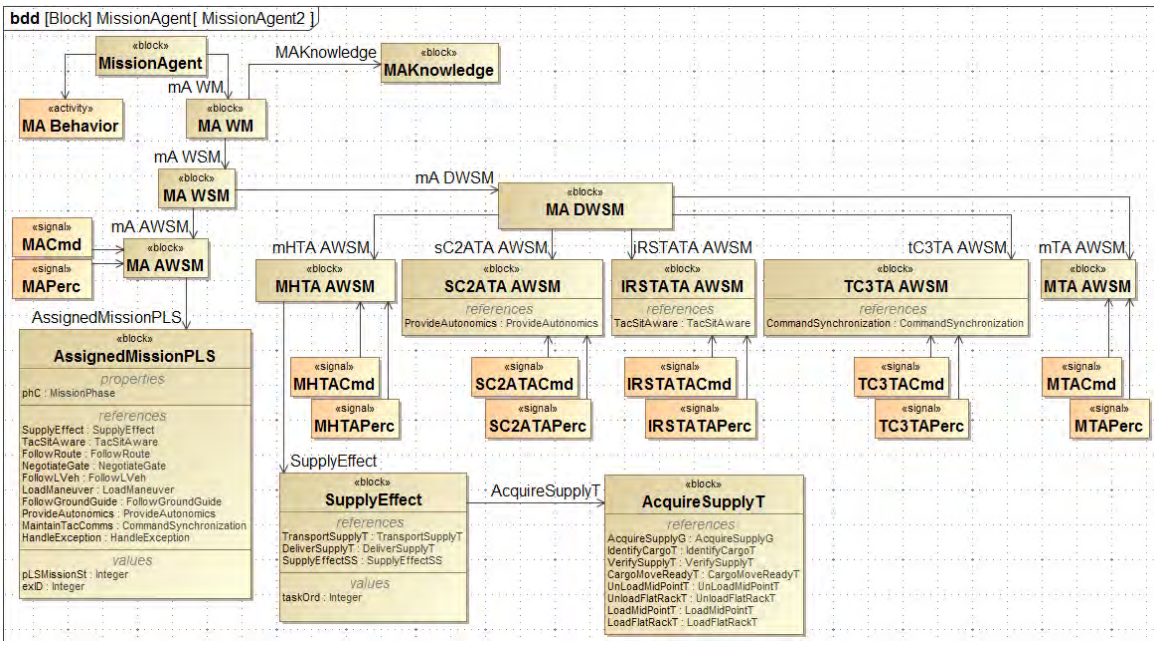

Figure 61. Mission Agent Internal Composition

e. Mission Agent Behavior

The mission agent behavior is shown in Figure 62 as a SysML activity diagram. It follows the general agent behavior pattern of Figure 49. The mission agent AWSM along with each task agent AWSM are types as parameter nodes. Also types as parameter nodes are the commands and percepts that go between the mission agent and convoy commander and the mission agent and the respective task agents. These are the same command percept types as shown in Figure 61 and has used for IBD connector typing of Figure 60. The agent behavior is very generally defined as manage assigned and manage derived. Fundamental to both behaviors is the ability to determine current state and time as expressed in the constraint equations, and then, assess whether the current state constitutes sufficient progress toward achieving the assigned goals. Beyond that, the agent internal behaviors generally move data in and out via commands and percepts and then store, access, and update as needed.

The actual detailed behavior is non-trivial. The data in represents commanded goal assignments from superior agents or status of current goals from subordinate agents. The mission agent would have to know specifically how to derive the needed goals

assignments and what would constitute sufficient progress toward achieving the goals. If sufficient progress was not achieved, the MA would have to derive a plan to fix and notify the convoy task commander. However, at this stage of conceptualization, it is more important to understand the data; that is, if the data is sufficient, an algorithm can likely be designed to achieve the detailed behavior. Important to the system conceptualization, is what data needs to be understood and how is this data to be acquired. Little would be gained by trying to capture behavior design in a detailed activity diagram at this stage.

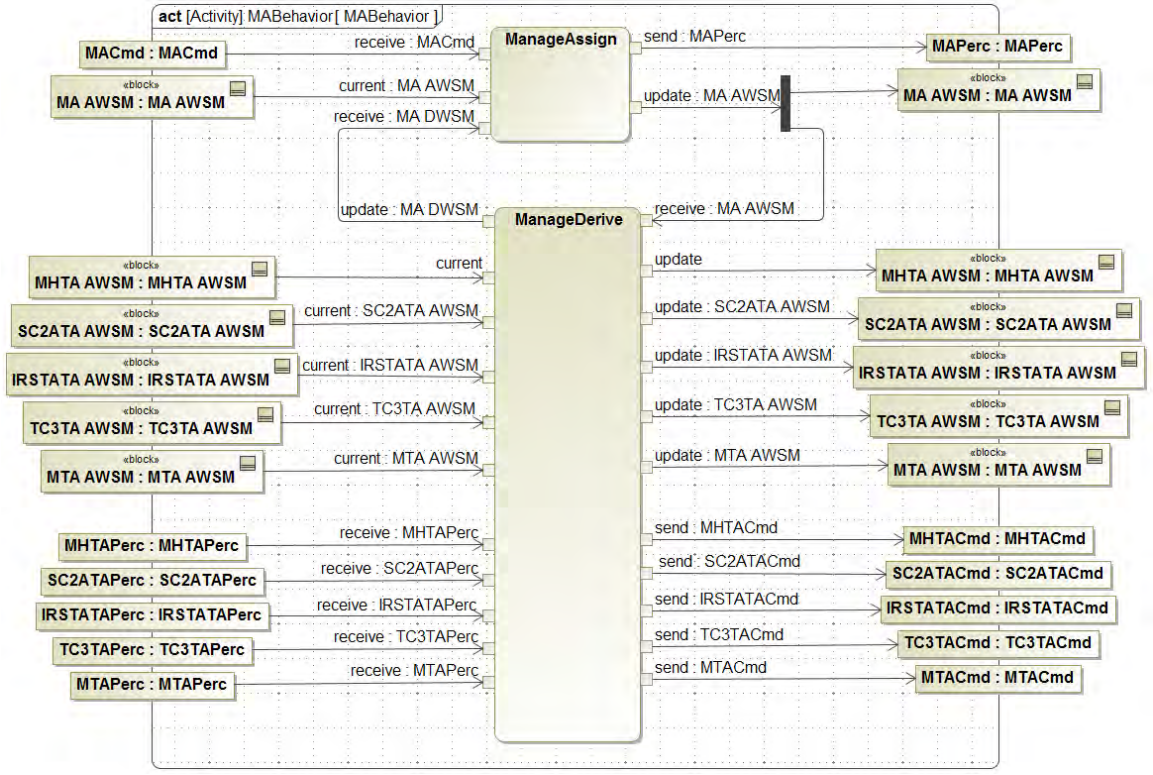

Figure 62. Mission Agent Behavior

f. Mission Agent Integration

The five model elements with some key relationships are shown in Figure 63. Together they provide the conceptualization of the mission agent. The full attribution of the mission agent assigned desired trajectories and related logical object hierarchy will not be completed until the lower level agents are elaborated. Since the lower level agent models are elaborating the model elements of the higher level agent, it will be attributed

as the lower level agent goals are defined. There is point in the lower level elaboration that goes beyond the concern of the mission agent (i.e., defines detailed goals below its assignment), but that detail is not viewable at that level of abstraction and can ignored by the mission agent.

Beginning with the assigned mission and desired trajectories, an agent with related performer context objects must be identified in the logical object hierarchy. The performer and context objects constitute the state space used to assign mission goals. The agent has an internal composition that includes behavior and world state models, an AWSM and a DWSM. The assigned mission is allocated to the AWSM and allocations or derivations of this AWSM is assigned to its DWSM, which is composed of the lower level agent AWSM models. The agent interacts with lower level agents and communicate goal state via commands and current state via percepts. The agent behavior manages this communications and compares goal state to current state to determine the correct course of action.

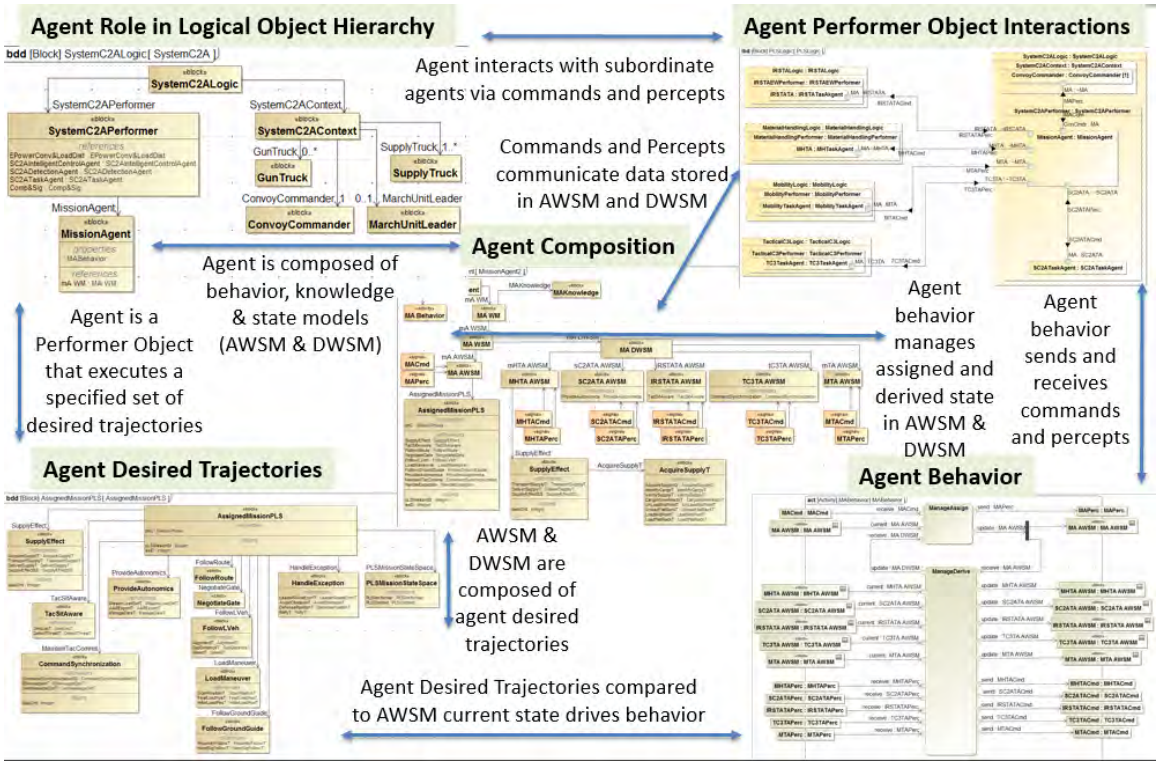

Figure 63. Agent Model Integration and Relationships

In similar fashion, each of the five model types are created for the lower level agents spread across the five performers. Unlike the mission agent, these agents detail the trajectories and link to goals until the interaction with controls and sensors are defined. As the goals are elaborated, it is more difficult to fit all the information on a single diagram, but they are integrated in the model and can by starting at the higher level and elaborating the upper trajectories. There also is an iteration of elaboration that needs to take place between the assigned desired trajectories, the logical object hierarchy and the agent composition. This iteration is not obvious in a sequential presentation. A detailed thread for Material Handling and more brief and nuanced descriptions of the other performer agents will follow. A more complete set of diagrams for each performer agents is included in the Appendix organized by top level performer objects.

2. Material Handling

PLS material handling has the components required to load, unload and transport supply as required by the mission. Delivering supply is the primary mission effect of the PLS.

a. Material Handling Assigned Mission

The next level of decomposition of the Supply Effect task from the MA assigned mission as shown in Figures 59 and 61, is shown in Figure 64. As was mentioned previously, has three top level desired trajectories, one of which, Acquire Supply has seven next level trajectories. Transport Supply and Deliver Supply have two and seven desired trajectories, respectively. Supply Effect has a State Space which is selected from the overall assigned mission state space and includes all the material handling performer objects and selected context objects relevant to material handling. The desired trajectories execute in sequential order: Acquire, Transport and Deliver.

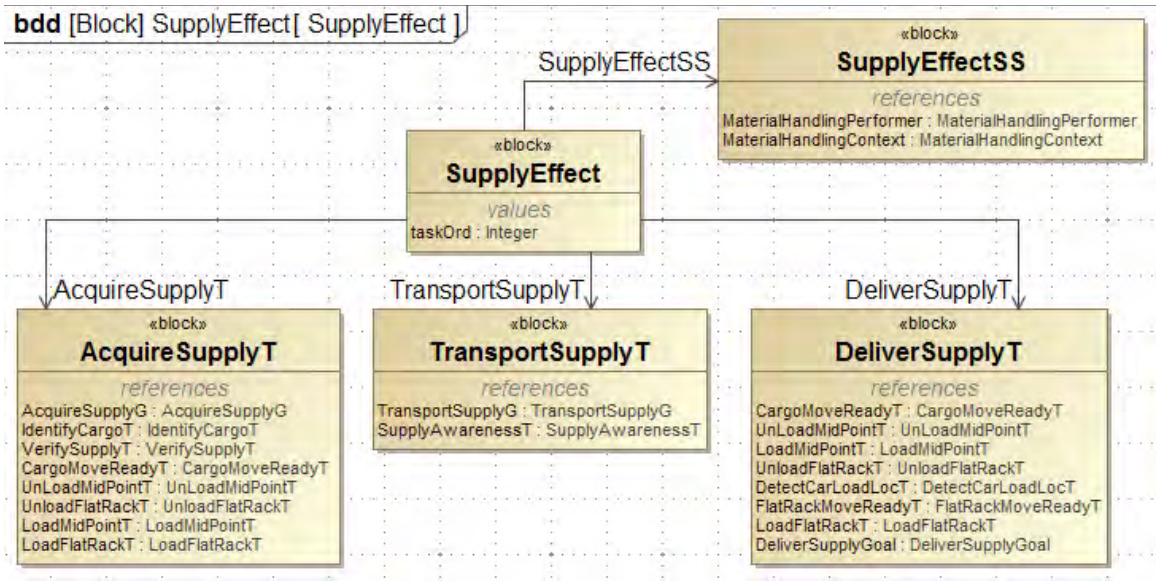

Figure 64. Supply Effect Task

The seven desired trajectories for Acquire Supply are shown in Figure 65. Each of these in turn has at least on supporting desired trajectory. Acquire Supply's mission phase is preparation. The task order is sequential as follows: Identify Cargo, Verify Supply, Cargo Move Ready, Unload Midpoint, Unload Flat Rack, Load Midpoint and Load Flat Rack. Cargo Move Ready has two levels of supporting desired trajectories. In general the system has to locate the cargo and make sure that it is the proper relative position to inspect the cargo content or load the cargo, verify that the cargo content matches the manifest, verify that the ground can support proper loading and that the cargo itself is ready to be loaded, and finally to load the cargo.

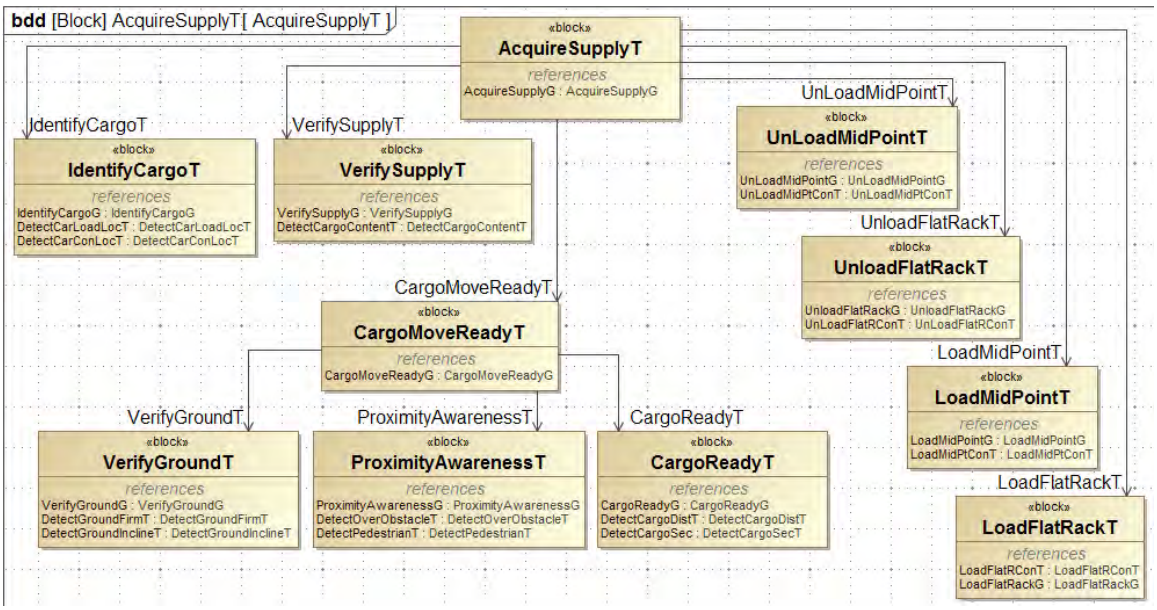

Figure 65. Acquire Supply Desired Trajectory

The cargo move ready goal is shown in Figure 66. It is typed as an achieve goal. It inherits all the goal properties from Figure 42 as shown in the Cargo Move Ready Goal block. Time goal measures are straightforward value assignments. The concern for this goal and all the goals in this modeling approach are the state-related measures. The state constraint equation is "redefined," a SysML procedure, and a new constraint equation defined that reflects the actual state for Cargo Move Ready. Once the actual state is established, the standard goal satisfactions constraints inherited can use the actual state (i.e., is the goal state satisfied; has the goal executed in the right phase and right order). Since there are three supporting desired trajectories, the cargo ready to move actual state, in this case, is expressed as a Boolean combination of those actual states. The state could be expressed as either a physical state or in terms of a value that can be compared to a goal target value of one.

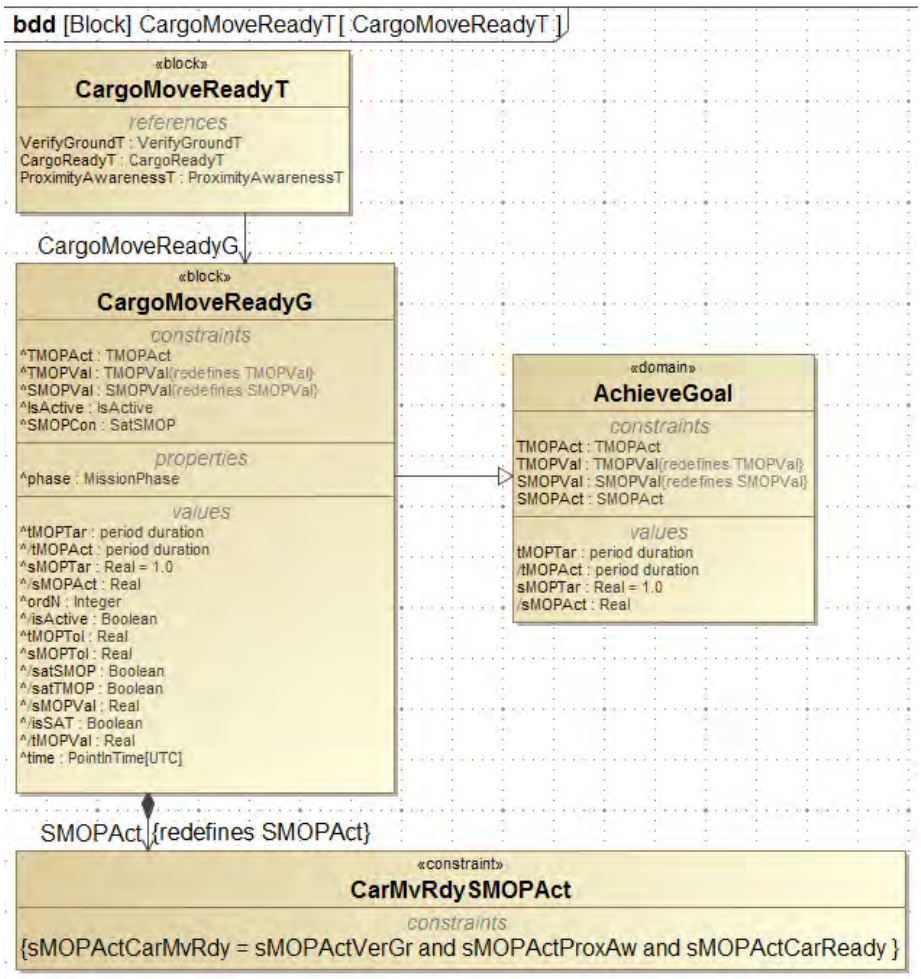

Figure 66. Cargo Move Ready Goal

The three supporting desired trajectories of Cargo Move Ready can be executed in any sequence. The Verify Ground goal is shown in Figure 67. The incline and firmness of the ground beyond certain limits will exceed the nominal design capacity of the loading mechanism. The cargo's actual incline and firmness relative to the ground or improved surface area will have to be checked against those limits. Verify ground has two supporting desired trajectories to detect actual incline and firmness. These trajectories in turn have goals with actual state constraint equation and supporting sensor trajectories. Their actual state constraint equations are expressed as a function of the sensor data. The point of detection trajectories is to detect incline and firmness events (i.e., achieve a state of knowledge). The ground is verified as being able to support loading when the actual

incline and firmness are within specified limits. This is expressed in terms of a difference function converted to a value expression. The other sub-trajectories, Proximity Awareness and Cargo Ready, are similarly elaborated and together are used to determine if the cargo is ready to be loaded.

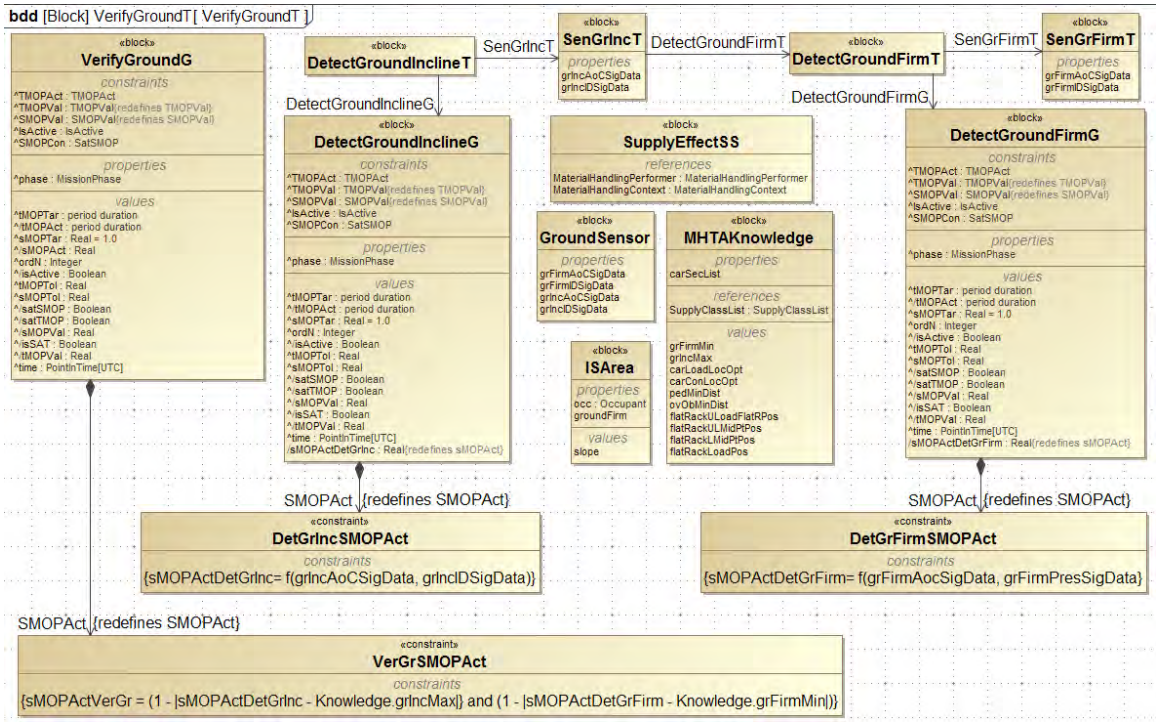

Figure 67. Verify Ground Goal

The overall state space for Verify Ground is shown in the center of Figure 67. It is drawn from overall Supply Effect state space and attributed as constraints are defined. The ground sensor has to provide the needed signal data to that senses the Improved Surface Area incline and firmness. The Improved Surface Area is a terrain context object. The Material Handling Task Agent (MHTA) must has the knowledge of the specified limits of incline and firmness used in the constraint state equation. The knowledge is part of the world model of the MHTA which is a performer objects. The MHTA Knowledge block has other knowledge required for state constraint equations form other desired trajectories.

Before it is determined that the cargo is ready to be loaded, the actual cargo must be determined and compared against the manifest. The human operator, or in this case an agent, is verifying that the supply that is supposed to be there, is actually there. This requires physical inspection of each supply item loaded on the pallet. The Verify Supply desired trajectory is shown in Figure 68. The actual state of the supply content is detected based on sensor data and compared against all possible types of supply with identified quantities. This actual state is then compared against the manifest using a distance function to determine the supply there, not there, and any unidentifiable items. This actual state is then compared to the goal target which is a one-to-one match of the actual cargo content to the manifest. Not addressed are the many nuances of goal tolerance. For example, missing a number of large caliber ammunition is probably much worse than missing an equivalent number of cigarette cartons.

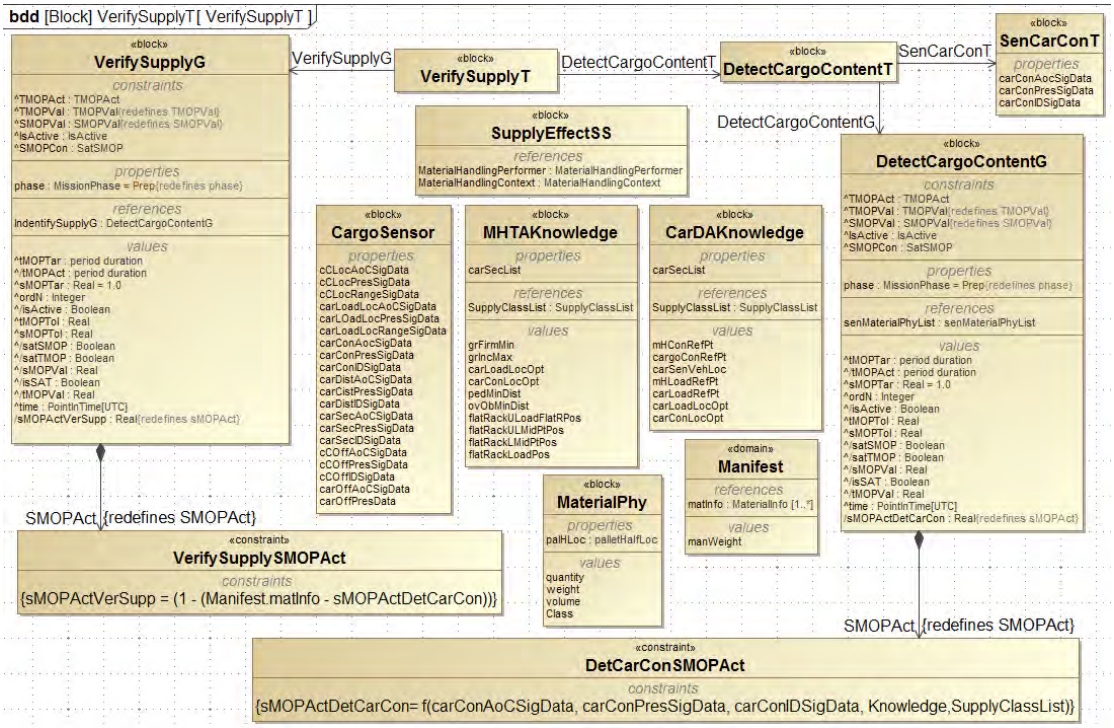

Figure 68. Verify Supply Desired Trajectory

The Verify Supply state space is shown in the center of Figure 68. It consists of the identified signal data from the sensor and the following references to supply content:

the actual content of the cargo, the manifest which is type of information object which in turn is a type of context operating resource, and the knowledge of all possible types of supply in both the Cargo Detection and Material Handling Task agents. The cargo sensor includes signal data from other Supply Effect trajectories that have a concern relative to the cargo such as whether is secured and the weight properly distribution from the Cargo Move Ready desired trajectory. This sensor data is shown in Figure 69 relative to the sensor reference trajectories and their associated detection desired trajectories. The reference trajectories refer to the commands sent to the sensor controller and may be simply be a control set point. The overall sensor state includes these trajectories along with the sensor signal data.

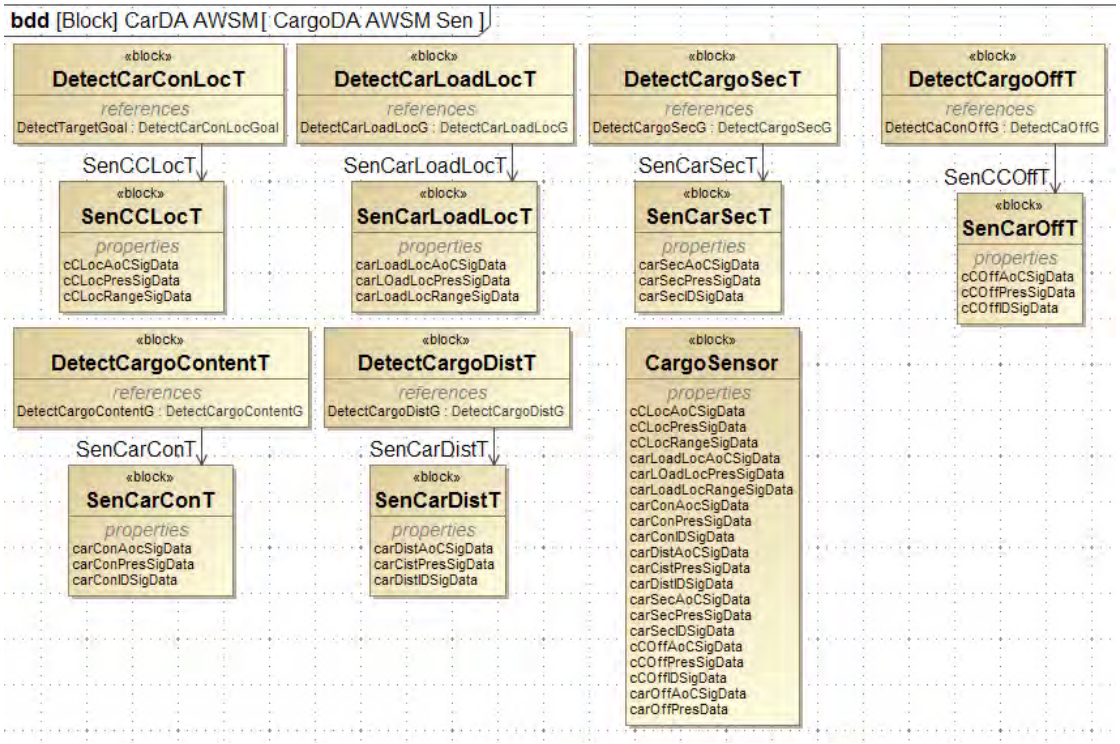

Figure 69. Cargo Sensor Reference Trajectories

Once the correct cargo is identified, the supply verified, and the cargo determined to be ready, it can be loaded. The most restrictive loading condition occurs with a minimum length to load. In this case, the flat rack is unloaded to a midpoint of the cargo bed, then the truck moved out while holding the flat rack position, and then the flat rack

is lowered to the ground. Some external supply agent then loads the pallet with the cargo onto the flat rack, and then the loading process proceeds in the reverse order. This is shown as the top level desired trajectories in Figure 70. Here again, there is an optimum flat rack position relative to the midpoints, vehicle location and the ground location. In each case the actual load state is calculated as a difference function between the optimum position and the actual flat rack position. Note that the movement of the vehicle occurs between the MA and Mobility Task Agent once the MHTA notifies the MA of the midpoint position.

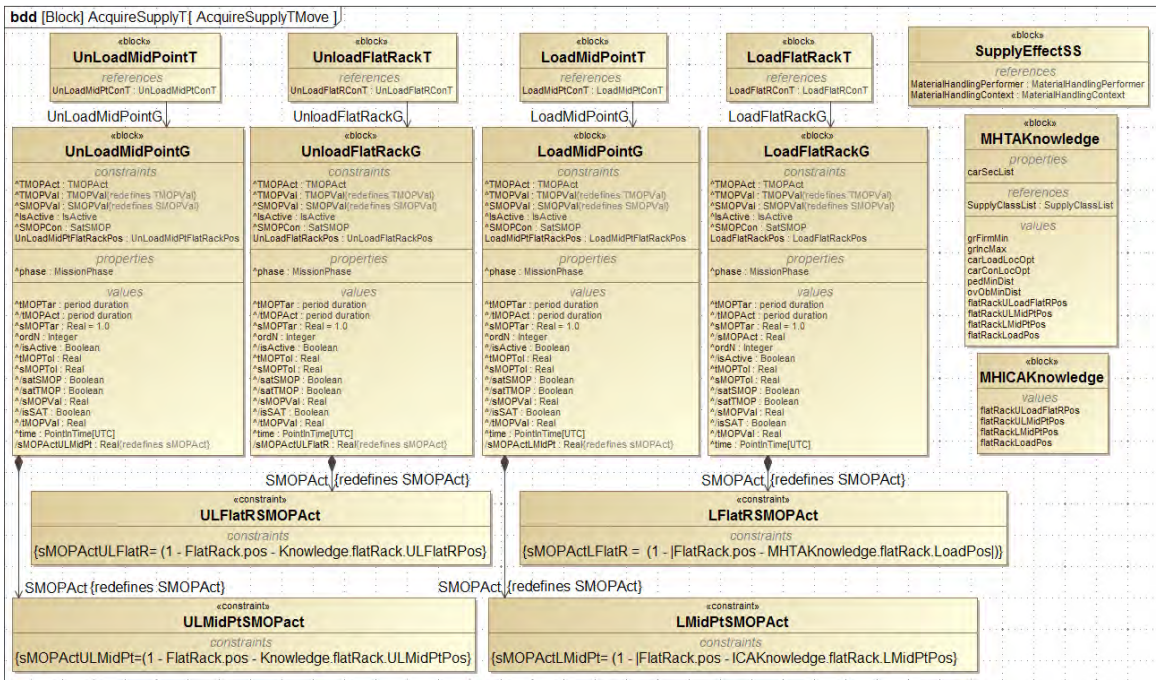

Figure 70. Load Supply Desired Trajectories

The loading state space is relatively simple, consisting of the optimum position knowledge and the flat rack position. As indicated in Figure 71, the flat rack position is a function of the feedback received from the four material handling controllers: Boom Control, Lift Control, Telescopic Control and Stabilization. This feedback is sent in response to the control reference trajectory or set point sent to each controller. The set point is unique to each load position and therefore as unique name. Once the flat rack is loaded with the cargo and the supply meets the target goal relative to the manifest,

acquire supply is complete. The next trajectories of Transport Supply and Deliver Supply proceed in sequence. These trajectories and other Acquire Supply trajectories of Figure 65 can be found in the Appendix.

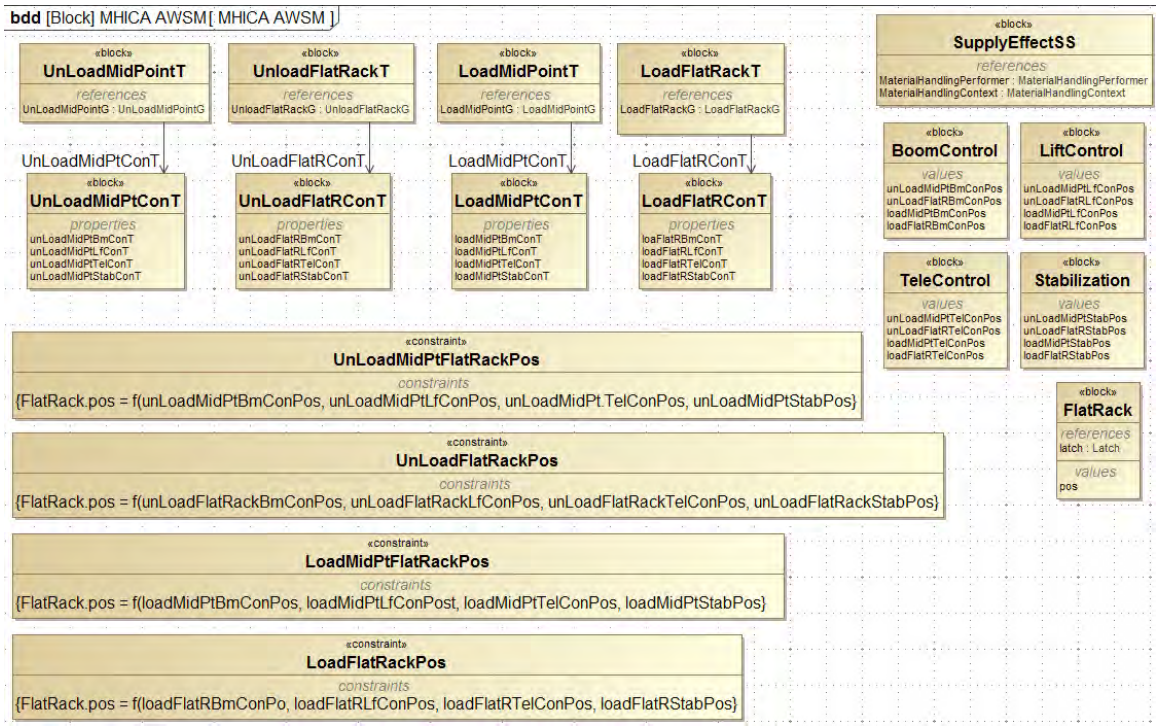

Figure 71. Intelligent Load Control Trajectories

b. Material Handling Logical Object Hierarchy

Portions of the material handling (MH) logical object hierarchy have been referenced in each of the desired trajectories previously discussed. The MH logical object hierarchy includes all those references minus duplication and with the performer agents specifically identified. It consists of a set of the performer objects and a set of context objects as shown in Figure 72.

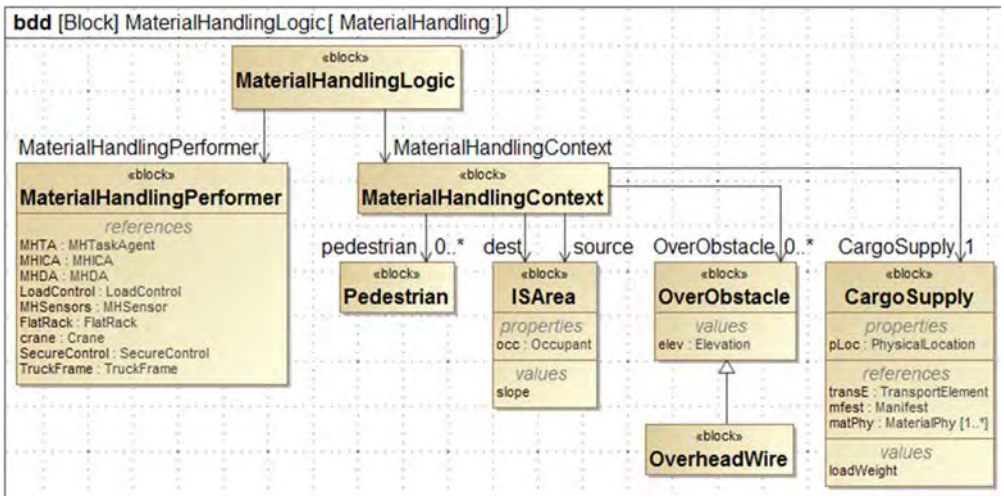

Figure 72. Material Handling Logical Object Hierarchy

The material handling performer objects are shown in Figure 73. There are six ALOs: one task agent, one intelligent control agent and four detection agents. Each detection agent corresponds to a sensor organized around the objects in the context that the sensor is to detect. The "sensor" is a conceptual abstraction to define the key state attributes and may be realized by a single sensor, multiple sensors, or be combined relative with other sensor abstractions. The sensor state is a combination of any control that determines its focus and/or area of coverage and the data itself. At this level of abstraction, the sensor converts some phenomenology to a data or cyber signal. Any interpretation of the context is done by the detection agent as function of the sensor signal data as identified in the constraint equation. Sensor signal data is identified by the information it will provide for use in the function, such as area of coverage, presence, range, identification, track, etc. The intelligent control agent controls for control objects: boom, telescope, lift and stabilization. These in turn control the plant (i.e., crane). The control objects are an abstractions of a controller, power amplifier and any feedback elements.

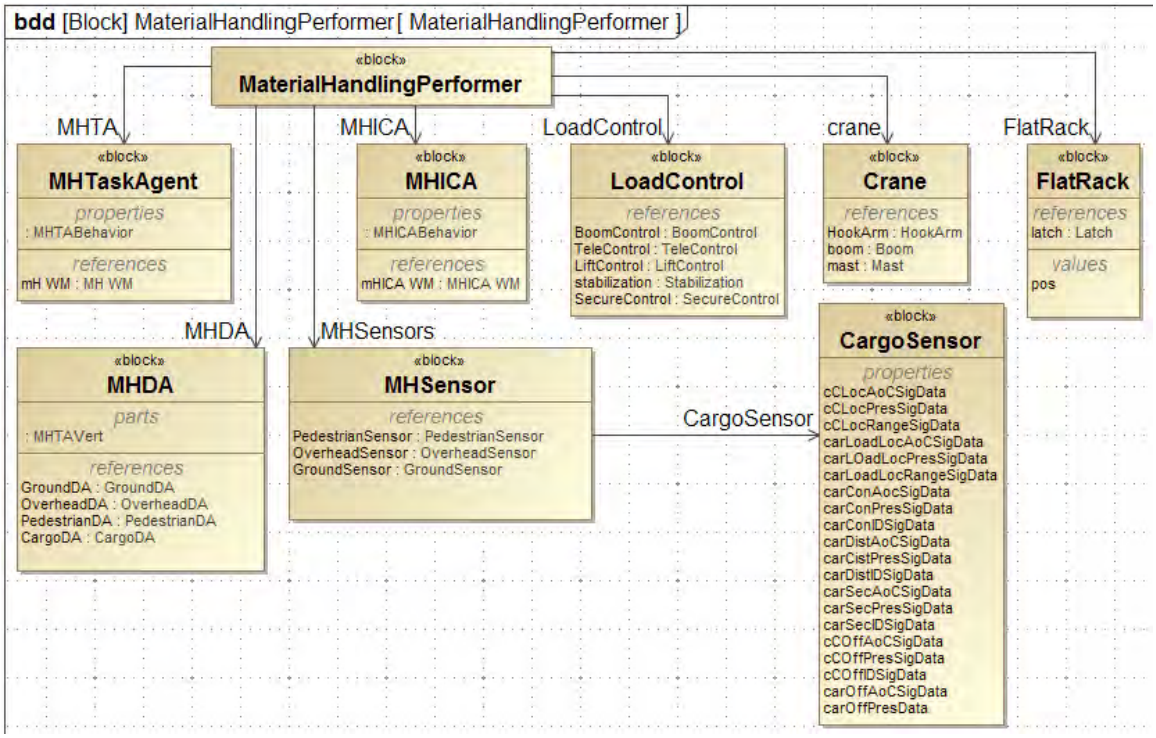

Figure 73. Material Handling Performer Logical Object Hierarchy

A key object in the context is defined as the Cargo Supply. It is the combination of transport elements, physical supply and the manifest as shown in Figure 74. The transport element consists of a pallet or ISO container and a means to secure the pallet (e.g., straps and hooks). The physical supply is the actual supply being shipped and can be any combination of ten classes of supply. Each of classes has sub-classes as shown for subsistence items. The manifest is a list of items that are supposed to be shipped. As was indicated before, the system must verify that the actual cargo content matches the manifest. For human operation, the crew or commander is accepting responsibility for that supply during transport. There is a third version of supply as part of the MH task and detection agent knowledge. This is used to for detecting the actual cargo on the pallet and includes a list of all possible types of supply. It is represented as a single property or list when referred to in the trajectory actual state constraint equations. However, each reference is actually a complex data structure and refers to the many data items listed in Figure 74 with specific identity and quantity. The constraint equations are framed as a weighted value composite of a many variable distance equation. For supply convoy, this

value composite is the key state effect from which to judge mission completeness. Mission completeness of 97% for example would be the percentage of items delivered relative to the manifest, with a presumed weighting of critical items over non-critical items.

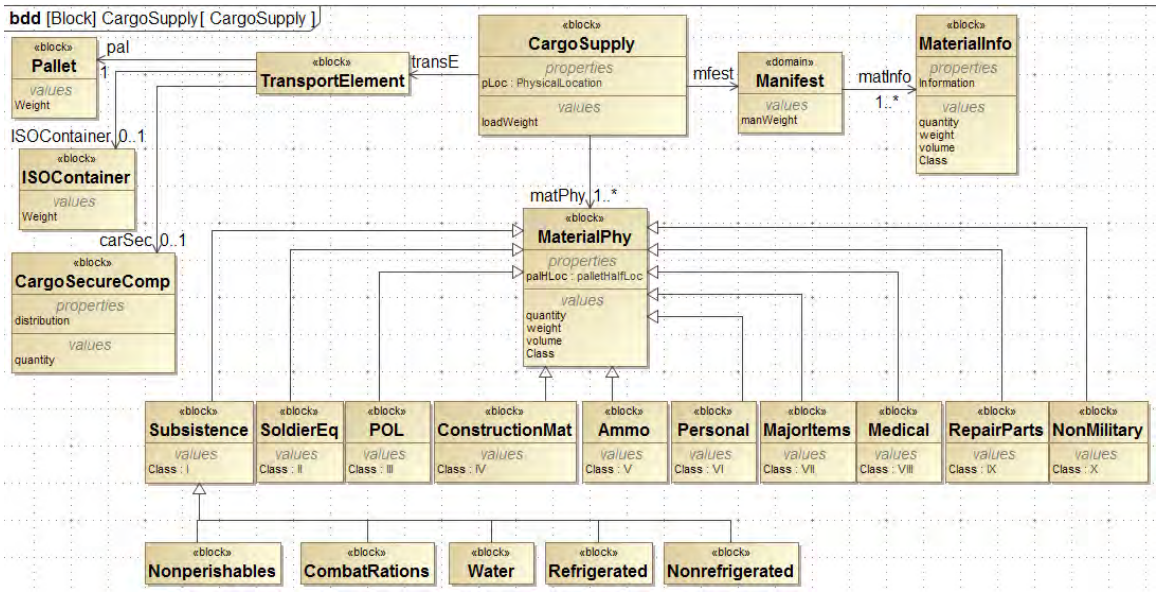

Figure 74. Cargo Supply Logical Object Hierarchy

c. Material Handling Interactions

The interactions of the agent objects, the detection agents and the sensor objects, and the intelligent control agent and control objects are shown in Figures 75, 76, and 77, respectively. The interactions again are indirect logical commands and percepts and follow the same pattern and naming convention as discussed previously for the mission and task agents. The detection agents share a name identifier and communicate one for one with a sensor: Cargo, Ground, Overhead, and Pedestrian. The intelligent control agent communicates with all control agents. This is required to insure a coordinated state and no contradictory direction. Though commands and percepts are sent to control and sensor objects, they are not managed as goal and current state within those objects. The commands are typically set points or reference trajectories as discussed previously. The response is feedback and or signal data as appropriate. Not that the mission agent

connected to the MHTA connected to ICA and DAs, and so on, are all connected. The entire system could be represented real estate permitting. Though the mission plan and associated desired trajectories have a certain sequence, the interactions can be concurrent and can be either asynchronous or synchronous. In this sense, the agent object executes more like an actor model than a finite state machine.

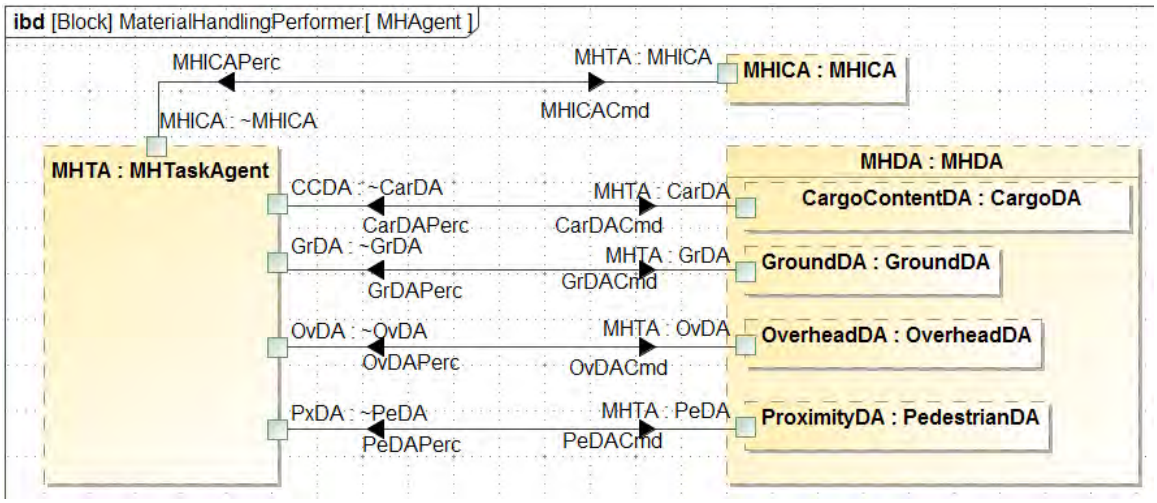

Figure 75. Material Handling Top Level Agent Interactions

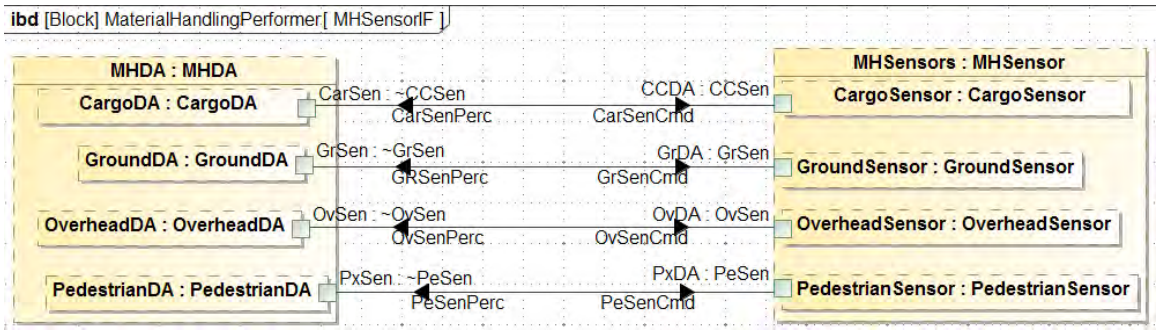

Figure 76. Detection Agent and Sensor Interactions

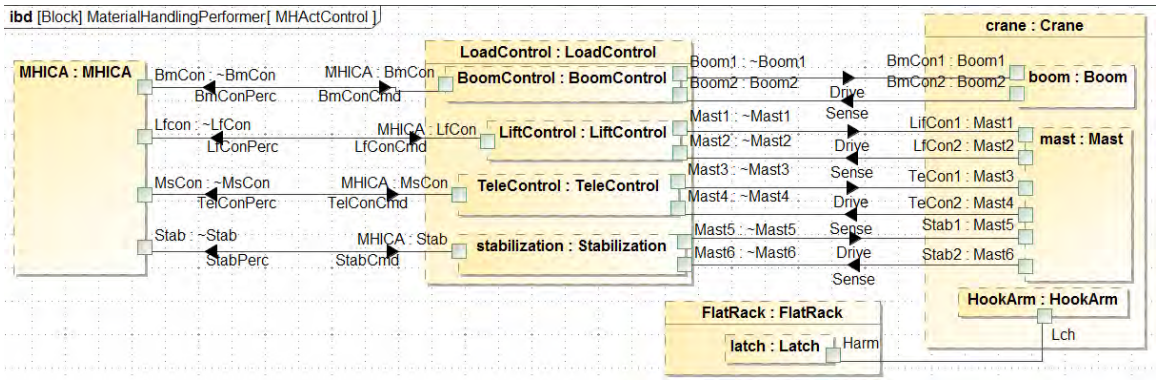

Figure 77. Intelligent Control Agent and Controller Interactions

d. Material Handling Internal Agent Composition

The internal agent composition of the material handling task agent (MHTA) is shown in Figure 78. It follows the agent internal composition pattern previously discussed and as presented for the mission agent. The MHTA assigned world state model (AWSM) has the supply effect trajectories as assigned from the mission agent. It has a derived world state model (DWSM) that is composed of the AWSMs of the subordinate agents: the intelligent control agent (MHICA) and four detection agents: Cargo (CarDA), Overhead (OvDA), Ground (GrDA) and Pedestrian (PeDA). The detail trajectories that are part Acquire, Transport and Deliver desired trajectories, are allocated to these various subordinate agents. The commands and percepts between the MHTA and mission agent and the MHTA and the subordinate agent as reflected in the IBD signals reference the respective AWSM. Trajectory goal states are communicated in commands and current or belief state is communicated in percepts. Also shown as part of the agent world model (WM) is the Knowledge that was required for the actual state constraint equations as elaborated in the assigned mission and desired trajectory decomposition.

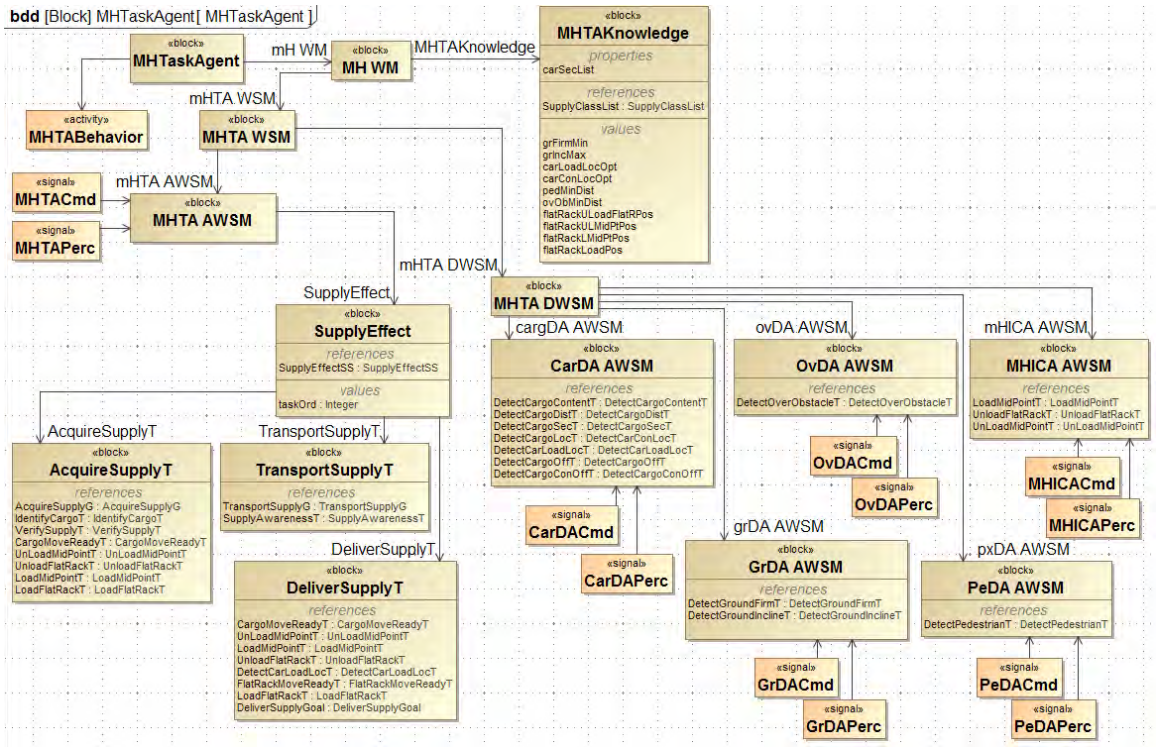

Figure 78. Material Handling Task Agent Composition

The CarDA is the most complex of the detection agents. It has seven assigned trajectories in its AWSM as shown in both Figure 78 and 79. It also has a DWSM that has a single AWSM for the sensor (CarSen) with the sensor trajectories and sensor data signals previously discussed. As is the pattern, the CarDA receives commands and sends percepts with the material handling task agent relative AWSM goal state and current state, respectively. Similarly, it sends commands to and receives percepts from the CarSen sensor. The sensor however does not manage these as goal state and current state, it manages it as an input (e.g., set point for control), and responds with signal data that the detection agent can interpret. The stored knowledge required includes optimum load location relative to the vehicle, the ability to recognize properly secured cargo, and an understanding of proper weight distribution. This knowledge again is used in the desired trajectory actual state constraint equations. The other three detection agents are similarly defined and included in the Appendix.

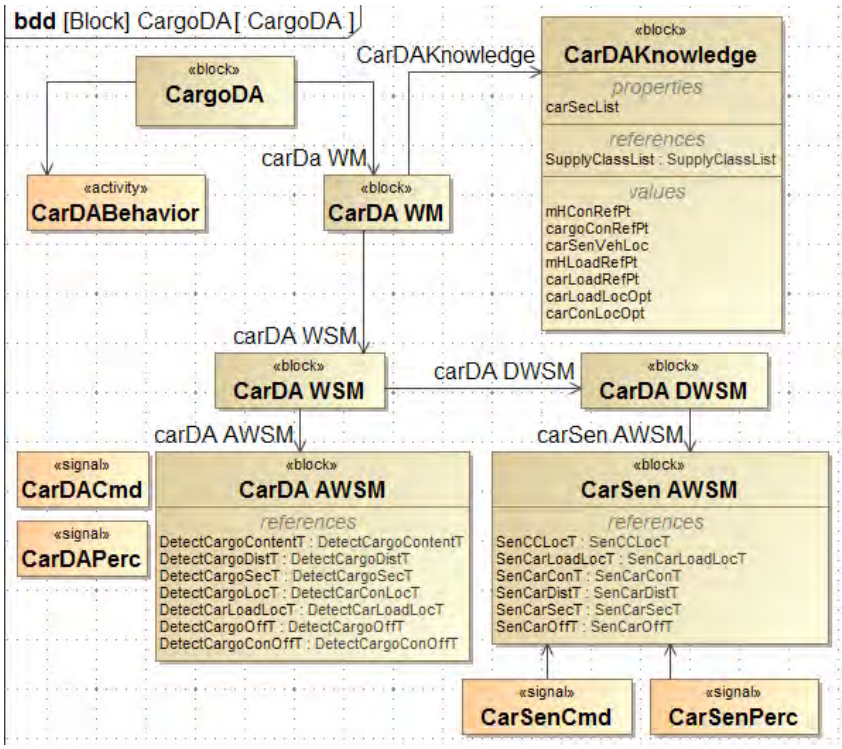

Figure 79. Cargo Detection Agent Composition

The material handling intelligent control agent (MHICA) is shown in Figure 80. Like any ICA, one ALO controls multiple control objects to insure an intelligent coordinated response. For each of its assigned trajectories: Load Midpoint, Unload Flat Rack, Load Midpoint, and Unload Flat Rack, the MHICA sends a control signal to Boom, Telescope, Lift and Stabilization control objects. Here again, these control objects see the commands as set points or reference trajectories and the percepts are just position feedback. The knowledge required by the MHICA is to know what the optimum positions are during loading and unloading. The MHICA assignment would reflect this knowledge as a target but may vary the tolerance based mission considerations.

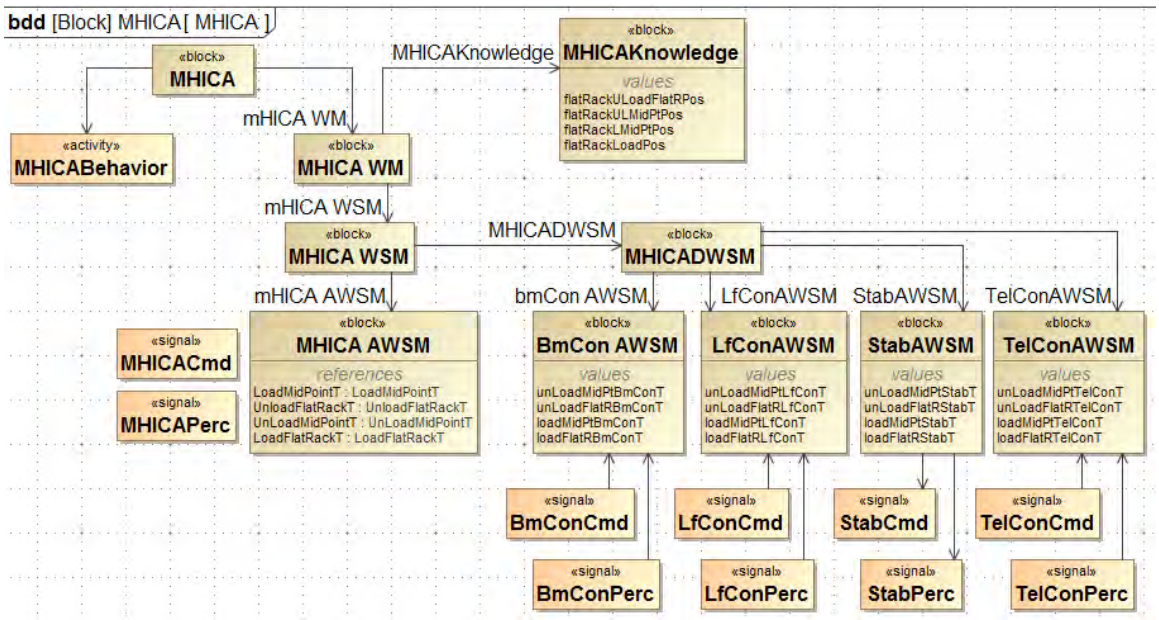

Figure 80. Intelligent Control Agent Composition

e. Material Handling Agent Behavior

Part of each ALO internal composition is behavior. As previously discussed, each ALO has the same behavior at this stage of conceptualization. It must manage the interaction with its superior agent and its subordinate agents, store its assignment, derive subordinate agent assignments, and take appropriate action based on current state. The distinction between each ALO is in the data or knowledge from which it bases its behavior. For the MHTA as shown in Figure 81, the commands and percepts with the mission agent type SysML parameter nodes and flow into an out of pins to manage assignment. It is similar for the commands and percepts for subordinate agents and manage derived assignment. These commands and percepts are the same signals that show up on the respective IBD and agent internal composition models. The respective AWSM as assigned from the mission agent, and the derived AWSMs of the subordinate agents, from the agent internal composition type SysML parameter nodes that are accessed and updated by the internal behavior. Again, the details of this behavior is not trivial, but more of a standard autonomous design task if so allocated. This behavior finds an optimum path through the state space given a goal state and a current state. The CarDA and MHICA are similarly defined in Figures 82 and 83.

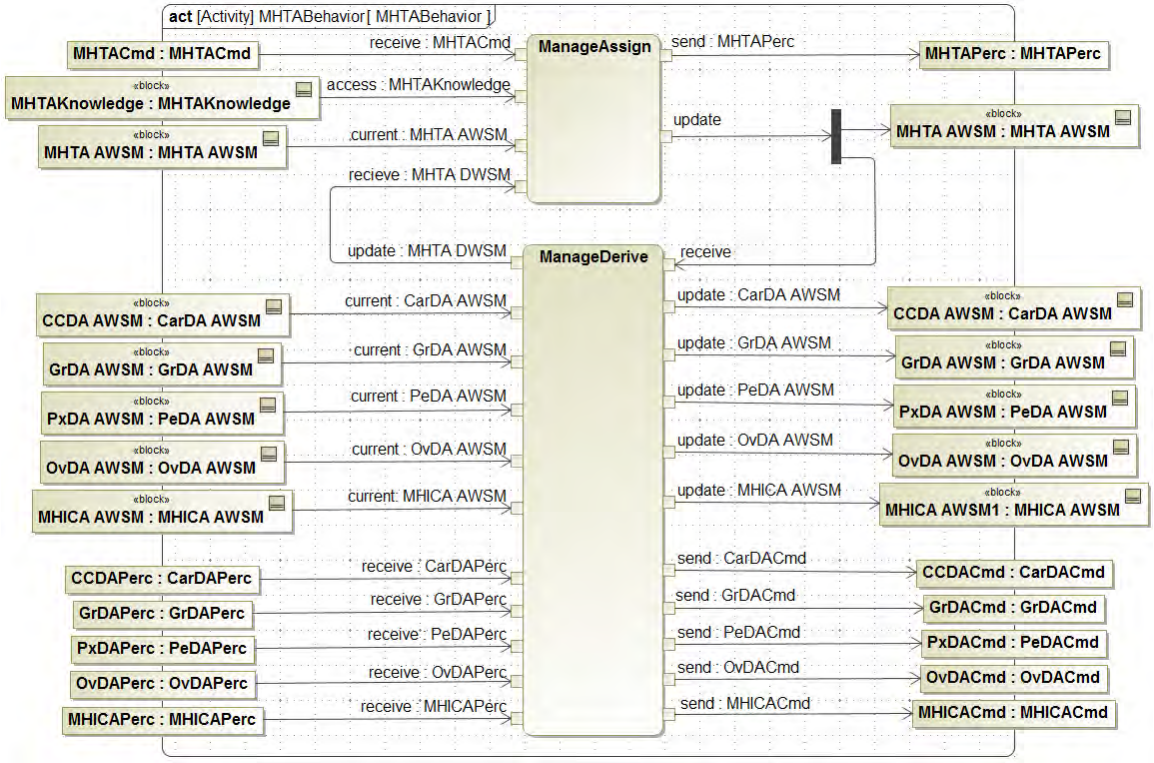

Figure 81. Material Handling Task Agent Behavior

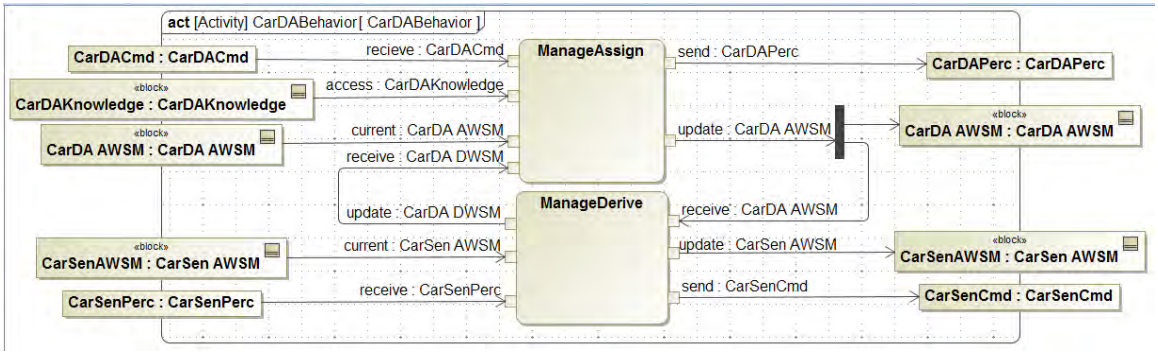

Figure 82. Cargo Detection Agent Behavior

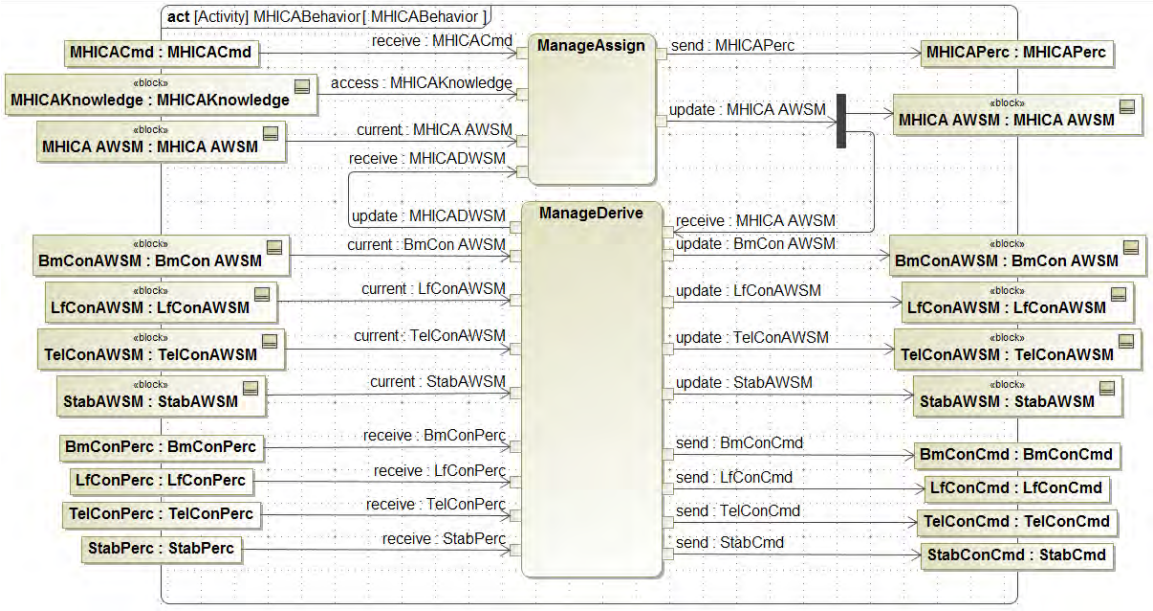

Figure 83. Material Handling Intelligent Control Agent Behavior

Each material handling diagram type has relationships to one or more of the other diagram type as shown for the mission agent in Figure 63. Though presented sequentially, much iteration needs to take place as the diagrams evolve. The mission agent integrates all the behavior of the performer objects through interaction with their respective task agents, but the goals are best elaborated within each performer object as allocated trajectories from the mission agent. All performer objects follow the same pattern as material handling and will be briefly be briefly summarized subsequently with the full complement of diagrams in the Appendix.

3. Tactical Command Control and Communications

The purpose of tactical command control and communications (TC3) is communications and information exchange with unit and other "friendly" elements. Actual mission command is performed by the mission agent. The TC3 performer object manages the conversion of information between the mission agent's world state model and the unit's standard messages. Some physical assumptions were made relative to the external information exchange. Namely, that voice interchange will be prominent, that it can be augmented with data, and that current interoperability data messages like the use

of radio procedure Prowords and tactical reports (e.g., spot, position), would follow the same formats as today. Separate voice and data channels were assumed and required response would have the same form as the input (e.g., if a voice command came in, there would be an acknowledgment in voice). This meant that the message state space would include whether it was voice or data in addition to its information content.

The radio or tactical communications logic had to be understood in terms of what is endemic to the identity of a radio and what is needed relative to human or intelligent behavior. The latter then had to be adapted to the general ALO, sensor and control pattern already described. This is best understood in the context of TC3 intelligent control agent (TC3ICA) and detection agents IBD as shown in Figure 84. For ease of modeling, a single transceiver was assumed, though logical this could be separated into separate channels and separate transmitter and receiver. Voice and data adapters provide data to a tactical network adapter. Adapters, like the transceivers, operate in both directions. TC3 “control objects” and TC3 “sensor objects” interface with the network adapter as opposed to the context like other performer objects. These objects sense and control content between the detection agents and intelligent control agent. For example, the tactical report sensor will recognize the message type as tactical coming from the network adapter, and provide it to the tactical report detection agent that can determine its content.

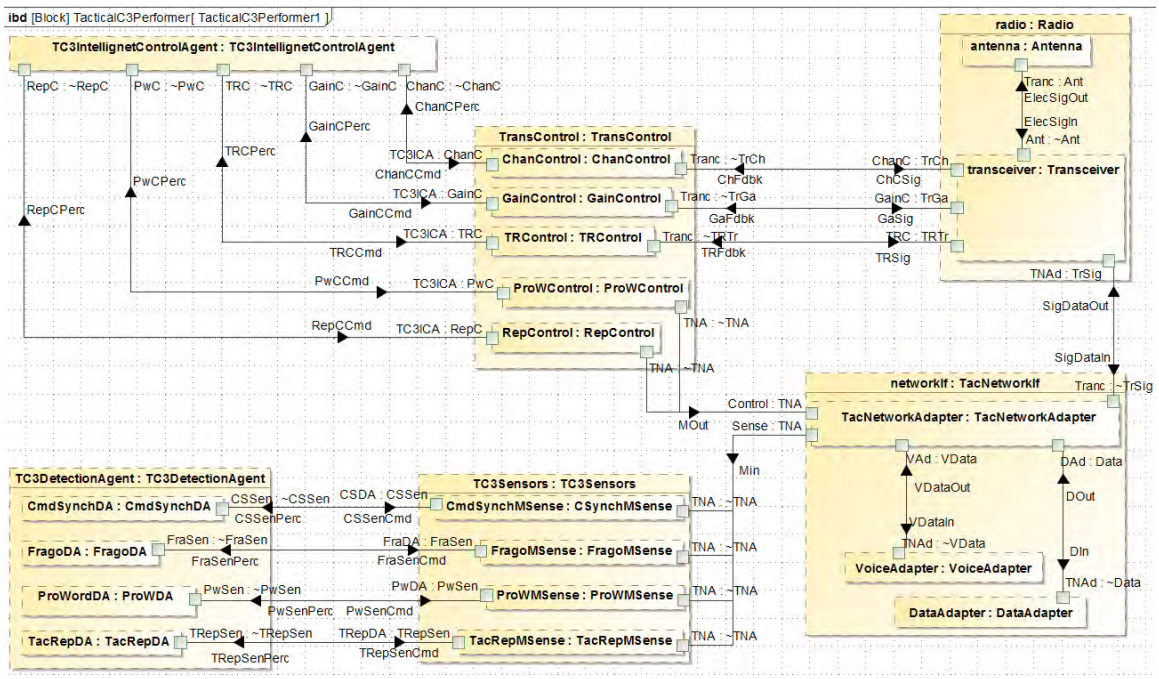

Figure 84. TC3 Intelligent Control and Detection Agent IBD

The data sensors and controllers, though somewhat artificial, can correlate to real logic that needs to exist to bridge between intelligence and data presentation. This logic would likely be combined as part of the network interface design and the terms would not likely be used. It does enable the general patterns as defined here to be used and the same set of diagrams can be generated. The TC3 task agent receives and generates the data in the same way as the other tasks agents. The TC3 performer object logical concept design is included in the Appendix.

4. Intelligence Reconnaissance Surveillance and Target Acquisition

The purpose of the intelligence reconnaissance surveillance and target acquisition (IRSTA) performer object is to detect threats in the context and to determine the geolocation of the system. The PLS as defined here does not have lethality, so it does not attempt to engage threats. It reports the presence of the threat and their geolocation trough TC3 so lethal units or systems (e.g., convoy supporting gun trucks), can take action. The desired trajectory of threat awareness is a maintain goal as shown in Figure 85. The state measure is defined as current versus actual to distinguish from an achieve

goal. The current state of threat awareness is the actual state of each threat that it can detect. In this case Armed Individual, Armed Vehicles, Improvised Explosive Devices (IEDs) or more precisely, their means of camouflage, concealment and deception (CCD), and "shots." Shots refer to the ability to detect a projectile that was fired and find its origin of fire. Its sensor would typically be physical realized as an acoustic sensor.

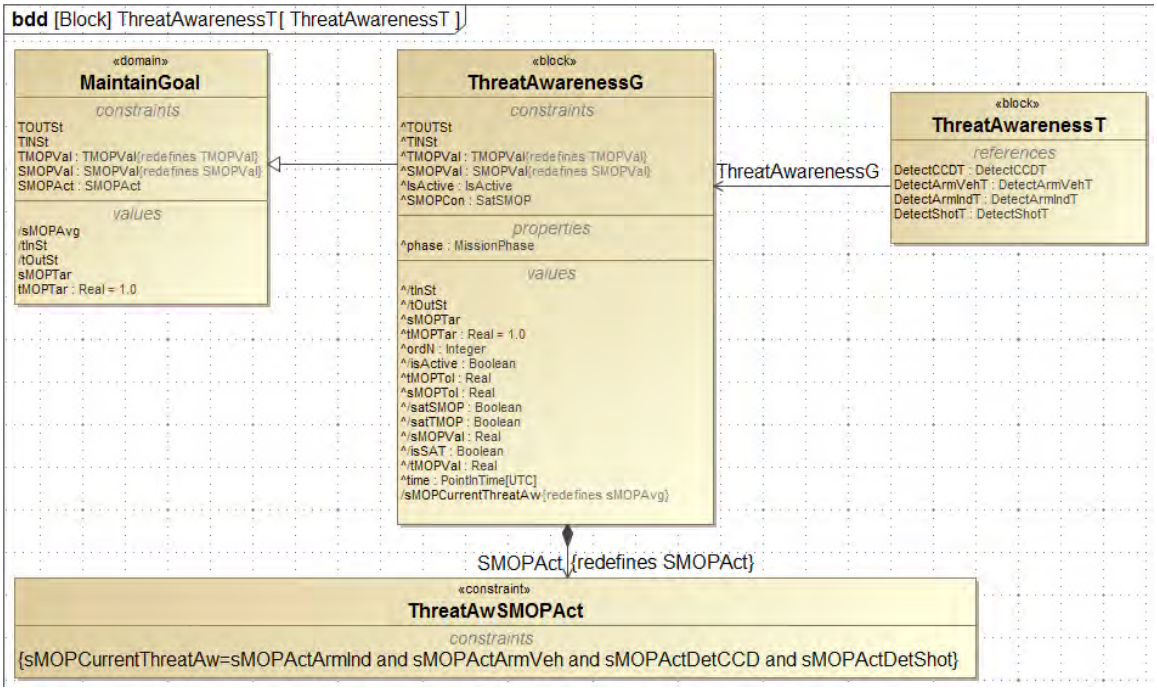

Figure 85. IRSTA Threat Awareness Goal

Each of these threat types has an associated detection agent and sensor per the agent pattern as indicated in the agent and sensor IBD shown in Figure 86. Also shown in the geolocation DA and sensor. Each detection agent uses the geolocation and range data from its sensor to calculate the threats geolocation. This location does not have to be precise as it is replacing a human operator's ability to estimate location. It also does not presume a physical solution such as the global position system. Note also that IRSTA is strictly a passive activity, it itself is not directly causing an effect or context state change. There is on intelligent control agent or controllers except what is needed for sensor control. Each of the detection agents are modeled to detect features about each of the threat types as shown in the desired trajectories within the Appendix. They again are not

fully classifying the threats, just characterizing them enough to support reporting and subsequent action.

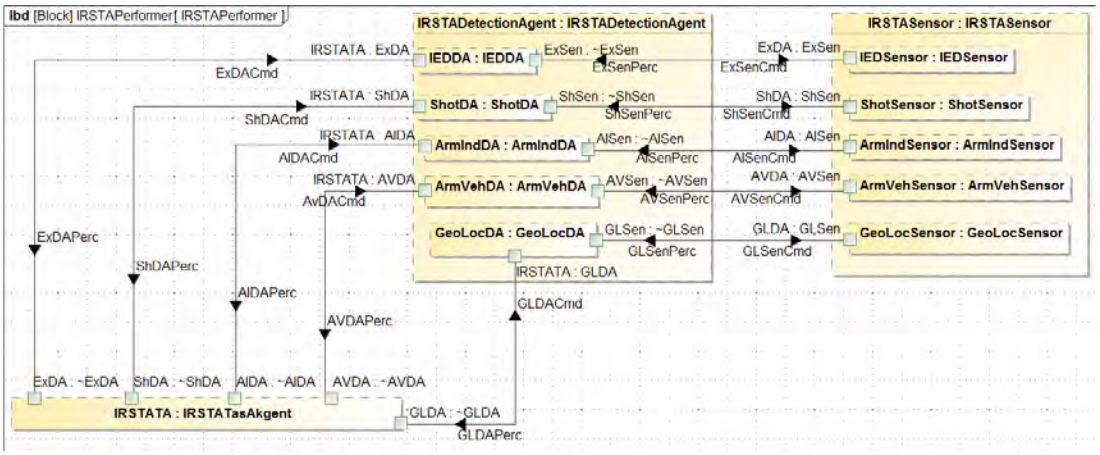

Figure 86. IRSTA Agent and Sensor Interactions

5. Mobility

The mobility diagrams have a different look and feel and some minor differences in information capture as explained in the Appendix. The mobility performer object is shown in Figure 87. The mobility task agent (MTA) has the conduct maneuver task as identified in the PLS mission. Its primary purpose is maintain following distance or time from its lead vehicle and is identified as a maintain goal. It is supported by several lower level desired trajectories: "Move To," Detect Obstacle, Detect Lead Vehicle, Detect Road Network, and Pass Route Point, as shown in Figure 88. In other words it has to locate the vehicle in front, move to a commanded position, avoid obstacles, follow the road and indicate when it has passed a route point (e.g., checkpoint). The full complement of mobility diagrams can be found in the Appendix.

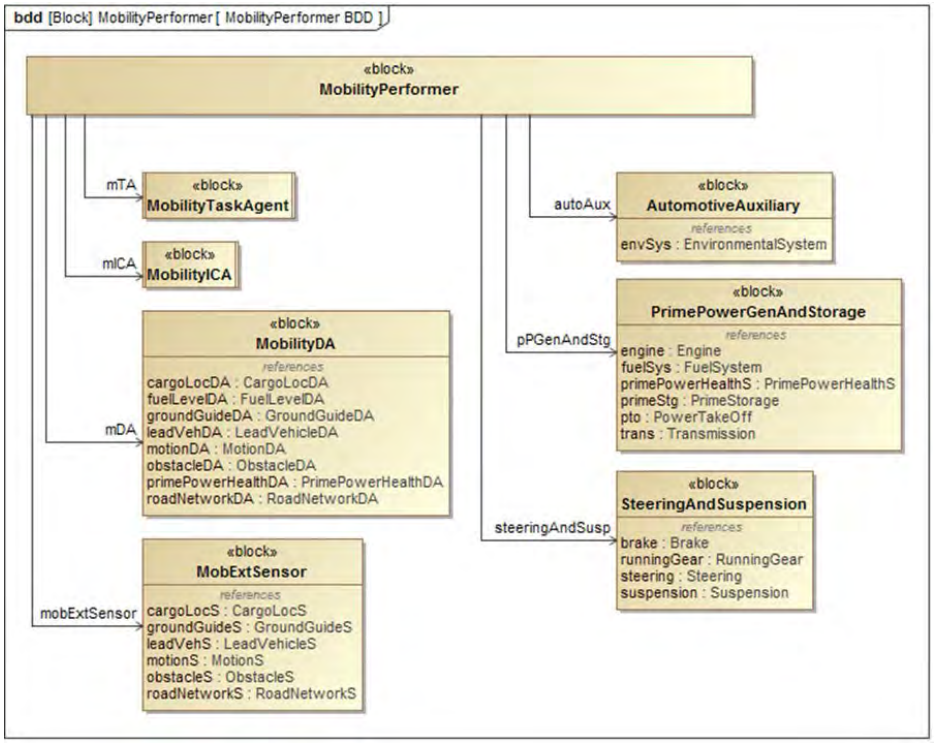

Figure 87. Mobility Performer Object

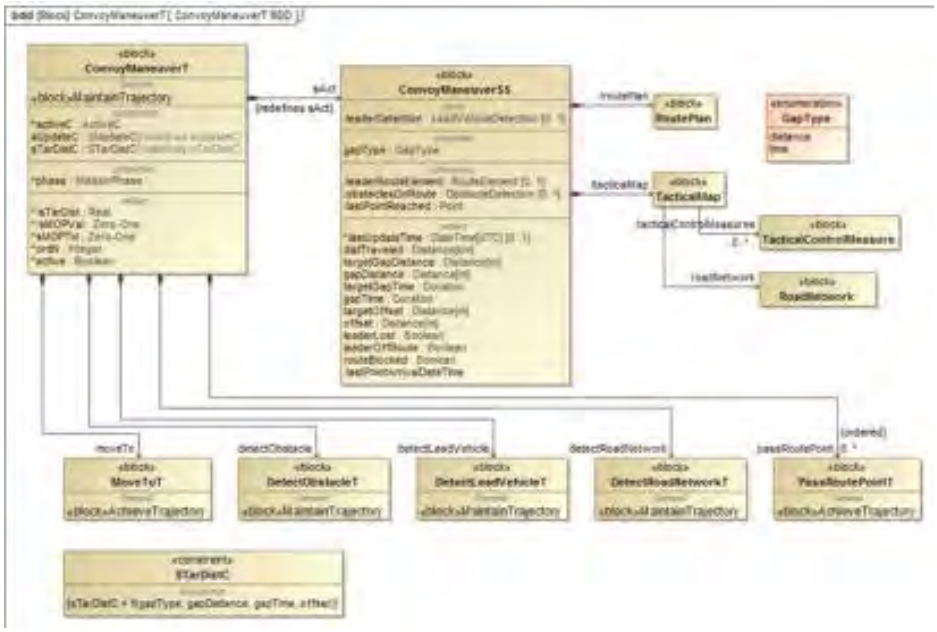

Figure 88. Conduct Convoy Maneuver Desired Trajectory

6. System Command Control and Autonomics

The top level desired trajectories of system command control and autonomics (SC2A) are: Mission Load Set, Mission Log, Play Mission Log and Provide System Computation and Signal Distribution. It must be able to detect that a mission is forthcoming. In the absence of a human operator, this requires that the system have some sort of permanent on state, at least on that can detect the initial command. From there it must be able to provide power to the system's computation and signal distribution for the required ALOs to operate. Ideally this would occur without the prime power of the system being engaged, particularly a combustion engine. A variety of mission data needs to be loaded (e.g., mission plan, map data, route plans, radio pre-sets). Each type of data must be routed to the appropriate ALO or ALO set. Once accomplished, the ALOs can behave and interact as previously described. During mission execution, the SC2A performer object captures and stores a mission data log. After the mission, the SC2A can play back or provide the data log for download. This is the equivalent of the PLS crew participating in an after action review.

The mission data log is captured by SC2A and its "sensor and "control" components. As with TC3, these are data sensors and data controllers rather than objects that interact with the context and would likely be part of some embedded computation or network logic. The SC2A sensor and controller objects detect events in the computation and signal logic of other performer objects and then captures and stores the event state. In this approach, all computation and signal objects are system resources. These resources can be viewed as being deployed in support of ALOs. A deployment like this would have to take place at design given the current state of practice. However, as practice evolves, this deployment could take place at run time (e.g., cloud computing), or even dynamically to adapt to varying resource conditions. Each ALO in each performer object has computation and signal objects deployed to it to support its execution. For example, the event sensor as shown in Figure 89 connects to the computation and signal logic of each performer ALO. It senses when an event occurs and passes the data to the Event Detection Agent, which then identifies the event and the relevant state for the SC2A task agent.

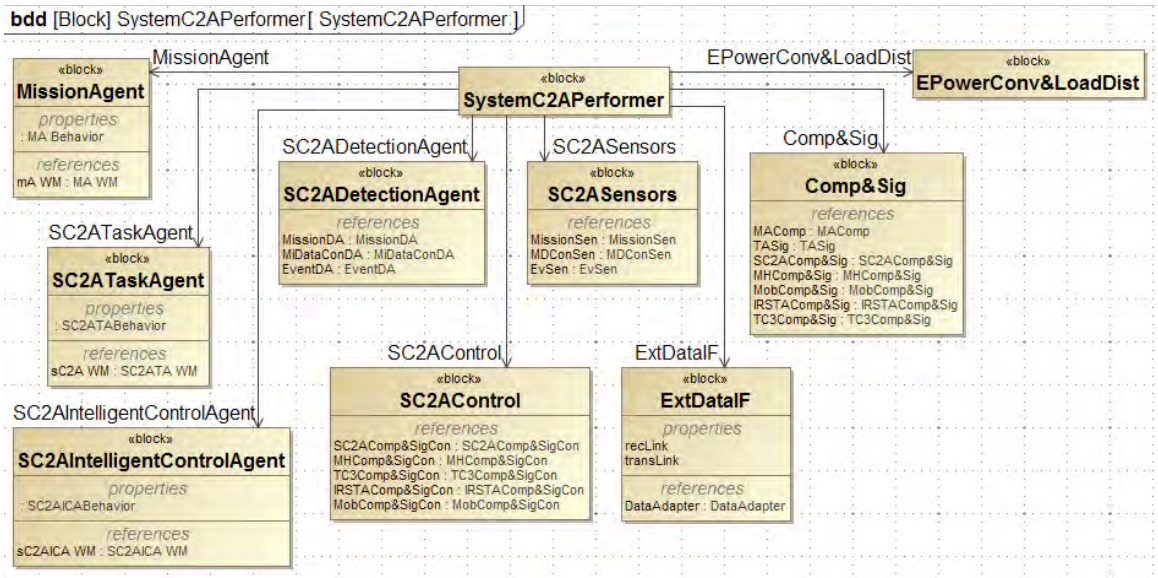

Figure 89. SC2A Performer Objects

As discussed previously, the focus of this research was on the horizontal interactions as isolated from vertical interactions or data transformation. However, to fully understand the SC2A performer objects, an illustration of the vertical logic for material handling (MH). As shown in Figure 90, each MH ALO has a computer object which represents a physical computing instance as, as opposed to a computer, that supports the ALO with the all the data transformations of a vertical stack. Once transformed, the data is communicated as signals between computer objects. The signal objects can be modeled as network or bus connections, as shown in Figure 91 for the detection agent computing, or as singular signal objects between each computer. As the computer and signal objects are aggregated into an actual computational and signal architecture, the execution time and throughput of the data transformation and distribution will constrain the performance of the horizontal interactions.

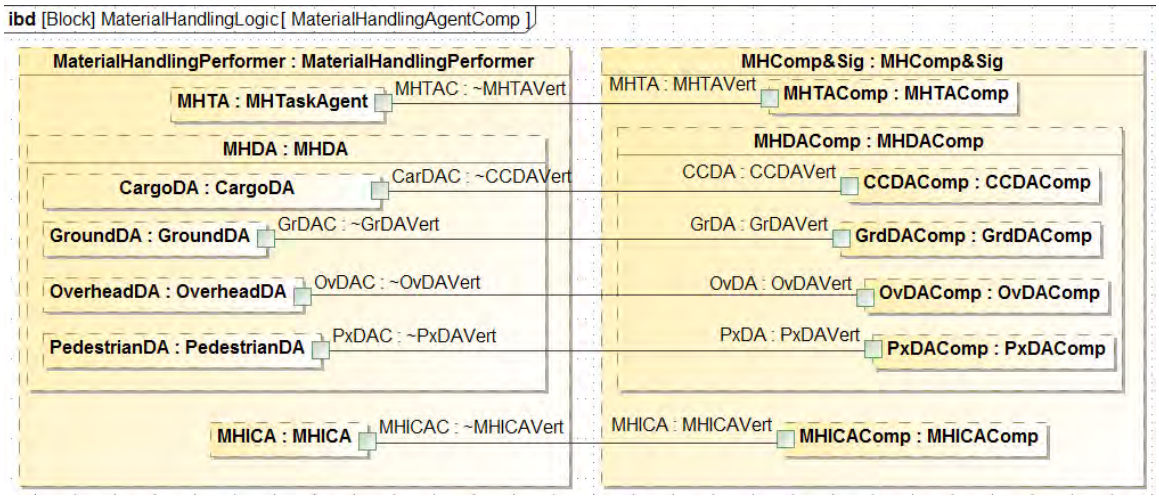

Figure 90. MH ALO and Computational Stack Vertical Interaction

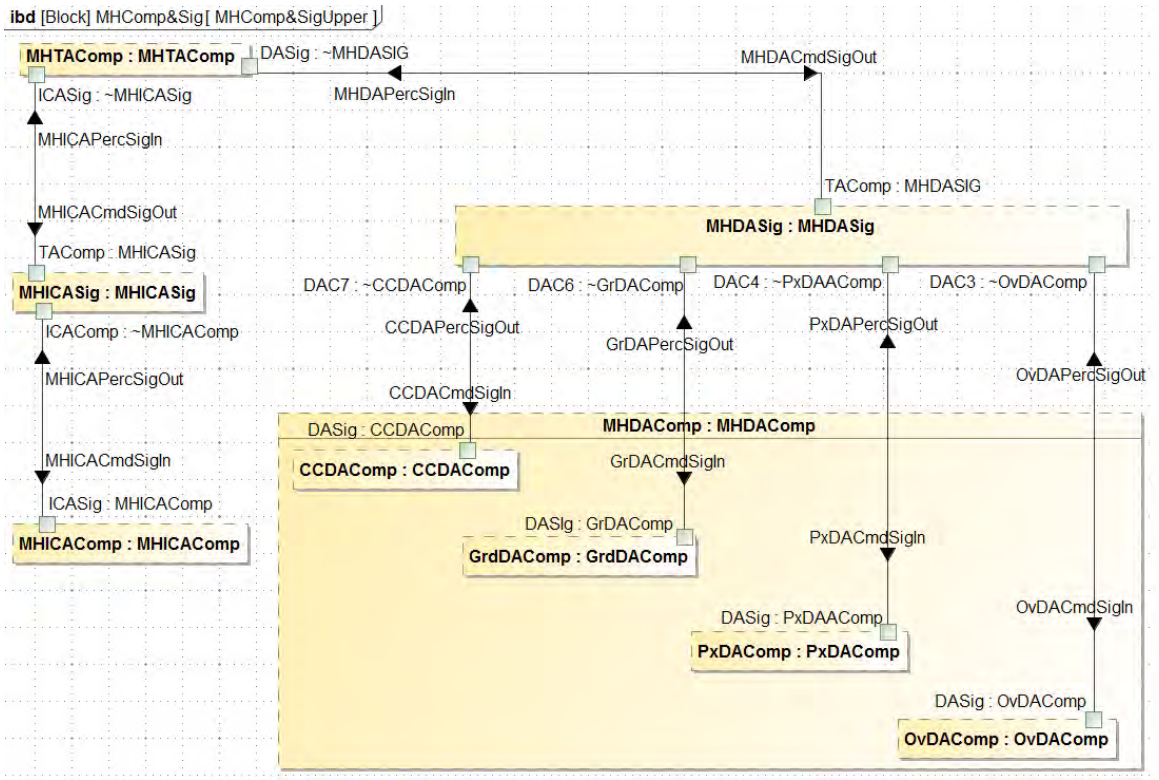

Figure 91. MH Computation and Signal Interaction

In summary, what was initially conceptualized as the mission agent state space of Figure 57, can now be considered to be a system architecture concept data model as shown in Figure 92. The system and its context have been elaborated in conjunction with the performer object elaboration and can be viewed as a set of structural entities and attributes. Performer objects can be extracted from the top level model along with any relevant context. As shown in Figure 93, the system has a material handling performer object which has a MHICA which has an assigned world state model that has several desired trajectories. One of those trajectories, Load Midpoint, has a goal with state and time attributes. Note that this is only part of the MHICA attribute set of state space. Similarly, as shown in Figure 94, the Cargo DA can be extracted from the top level set of performer objects and its state space identified. The context must also be elaborated to fully understand its state space. A mobile, goal directed, and context aware system requires both system and context definition to fully understand the systems data and signal attributes.

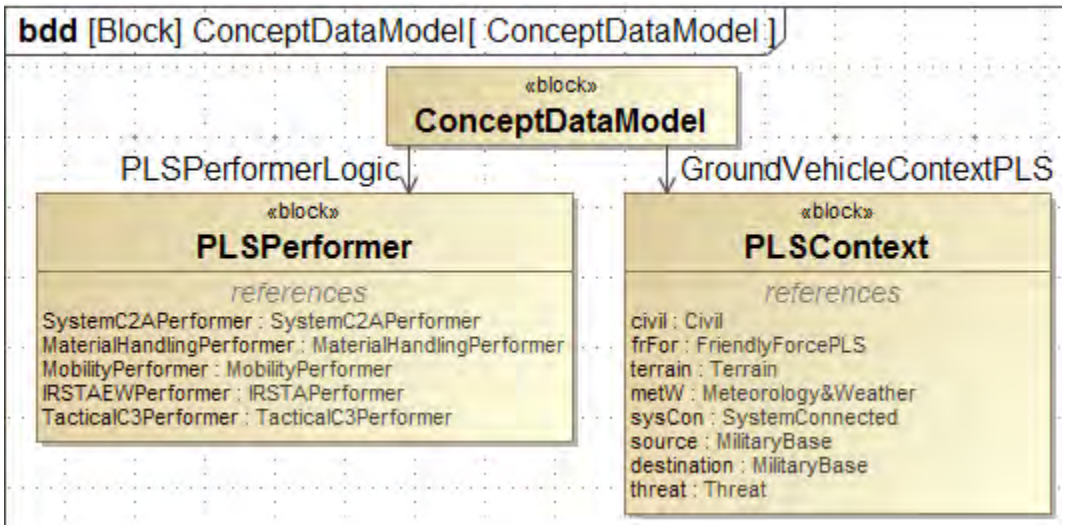

Figure 92. PLS System Architecture Concept Data Model

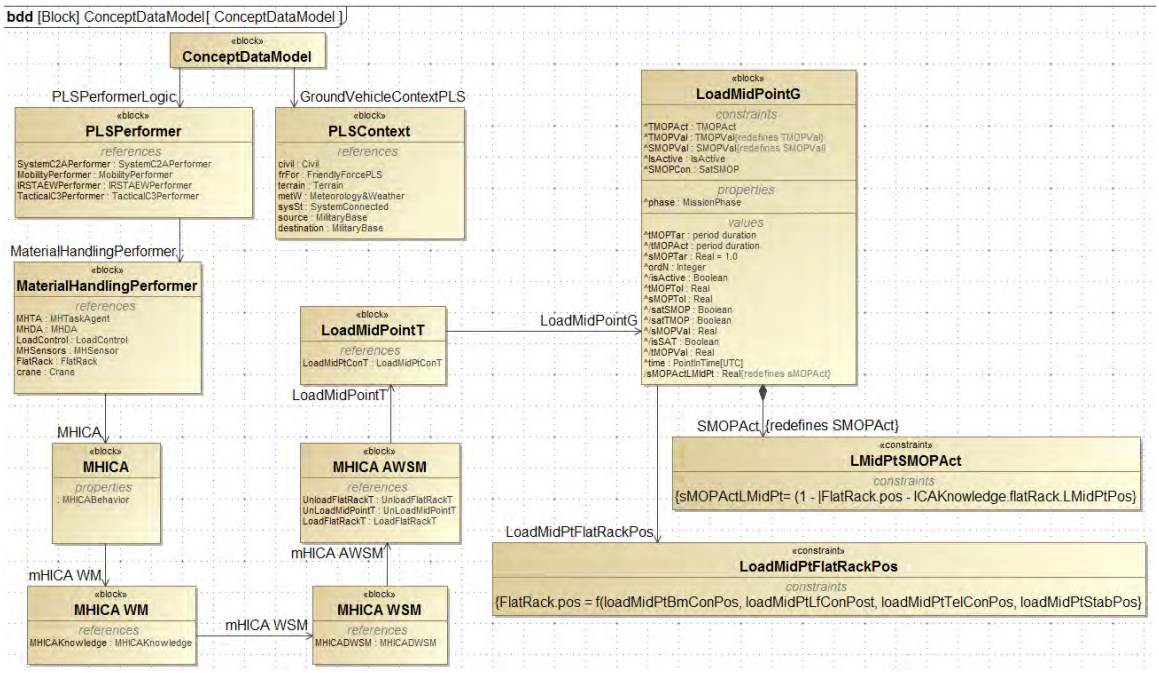

Figure 93. MHICA Concept Data Example

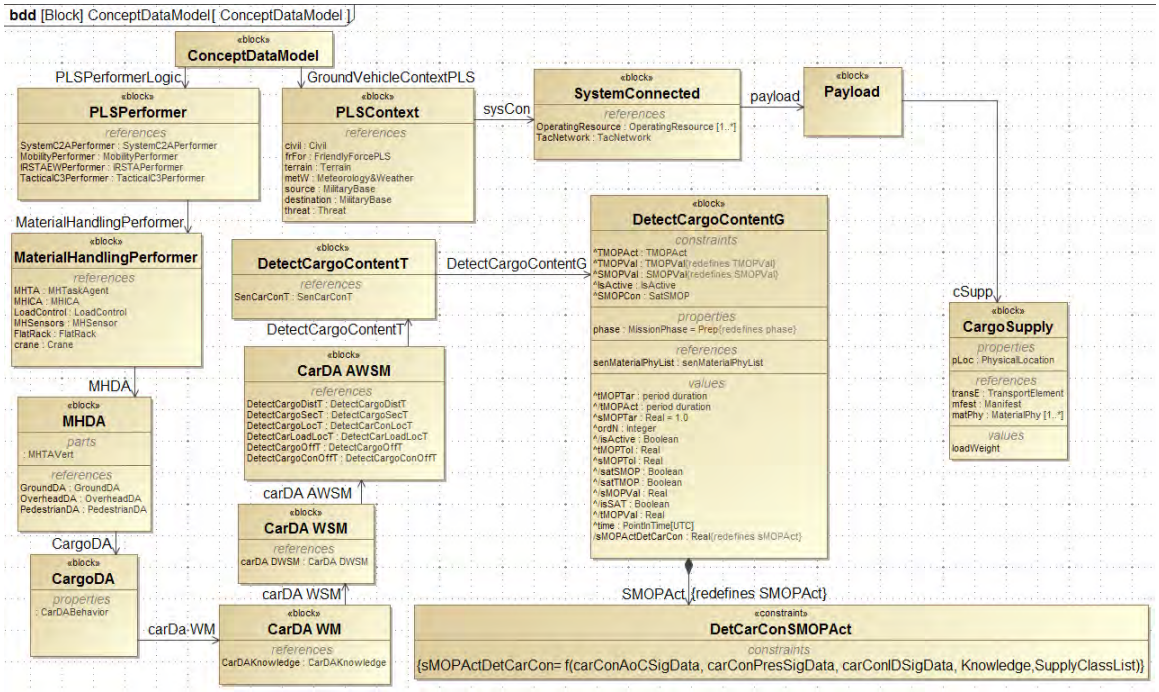

Figure 94. Cargo DA Concept Data Example

C. PLS MODEL QUALITATIVE ANALYSIS

The case study model was generated using the MCPS Concept DM2 and the supporting foundational concepts. In general, the five research questions posed have an affirmative response. Detailed responses are as follows:

Technological independence. The system's logical structure was modeled without specifying a technology solution with some minor exceptions. A ground vehicle has an identity relationship relative to certain physical abstractions (i.e., a ground vehicle has some means of propulsion). However, this is different than saying that the propulsion subsystem will be a diesel or combustion engine or electric drive. A PLS by identity has a loading mechanism. This perhaps can be something distinct from a crane with a boom, a stabilization system, etc. However, if some alternative exists, it could be equivalently expressed and linked into the upper levels of the material handling model. That alternative mechanism should have some means of being controlled and would integrate with the material handling intelligent control agent in the same way as the crane's control systems.

The "sensors" were framed to be realized by a device. They interact with the context and produce signal data to support an area of coverage, presence, range and bearing, identification, and tracking. However, a direct analogy can be drawn relative to "human sensors" that perform those same functions. When considering the state of technology relative to human capability, there are likely differences in precision and overall intelligent performance of the sensor in conjunction with the detection agent and task agent, but the combined allocation can be compared and assessed within a trade space. All the sensor and agent logical objects considered together can be allocated to machine, human or some combination thereof. When allocated to the combination, the ALO and sensor objects would aid in the derivation of the human to machine interface. The ALO machine realization can be either application software or the computational equivalent (e.g., a neural network).

The most significant constraint on technology independence occurs relative to a capability achieved by the system interoperating with some external component or

system, in particular the Global Positioning System (GPS), RF communications and mission data export and import. Though geolocation was abstracted as a desired trajectory, the supporting performer object correlates to a GPS abstraction except for teasing out the human and system interface. The system by itself cannot reasonably replace GPS with another technology. Similarly, some means of communicating between the system and the rest of the force is presumed to be very similar to the current military radio approach, with the added presumption of both a voice and data channel. Finally, it is also presumed that a smart system would benefit by the digital realization and transfer of major sources of data. This data includes mission plans and orders, communication presets, maps and mission logs. It would be required for an autonomous ground system and currently is often utilized relative to military aircraft.

Equivalent consideration to multiple forms of behavior. The major forms of behavior are: simple functional, state-based discrete event, continuous, and intelligent. A MCPS has a hierarchy of computational control with a CPS layer at its base. This layer is where discrete control interacts with continuous physical processes. The Intelligent Agent and Detection Agents form an intelligent behavior layer on the base CPS layer. They in turn are controlled by a higher intelligent layer of mission and task agents. Simple functional behavior is accommodated within the CPS layer and within specific ALOs as needed to support intelligent behavior. State-based discrete event behavior consists of information transformation within SC2A performer objects and to a lesser extent, TC3 performer objects.

Except for simple functional behavior which can be distributed throughout, the MCPS performer objects can be associated with a single form of behavior and associated computation. Specific objects within SC2A and TC3 transform information, ALOs accommodate intelligent behavior, and the CPS control objects accommodate direct discrete control of continuous behavior. These objects are similarly visible within the system concept model and can be linked to a specific computation units as machine instances or human roles. These instances and roles can be aggregated into physical solutions and the impact and contribution to mission and task performance tracked. All

forms of behavior are thus accommodated and can be specifically addressed relative to trade space assessment.

Integration of operational and system behavior. Operational task decomposition takes place in the context of a mission assignment with detail desired trajectories linked to a state space and goals. Specified goals link operational MOEs/MOPs of the behavior and are embedded in the model. This can be compared to use case analysis, which may look at similar behavior and goals, but are used to generate system functions, leaving the use case itself anecdotal versus part of the concept design. The embedded mission assignment is linked to METT TC variables via the associated state space that includes the necessary context objects and attributes.

Rather than directly mapping to system functions, desired trajectories are mapped to system performer objects or ALOs. More detailed mission or task decomposition takes place in the context of these ALOs and their decomposition. Further system logic decomposition is the result of the both logical structural decomposition as well as the details of the object behavior. The eventual decomposition reaches sensor and controllers which could be identified as system behavior. In addition, the interacting ALOs or the horizontal logic is distinct from the vertical logic, the latter can also be defined as system functions, to include computation and information processing. At this stage of conceptualization however, the logic as modeled can be considered to be all system behavior. If system performer objects are realized by human operators as physical solutions, a distinction of operational behavior may be useful for training purposes. With intelligent systems, distinct lines between operation and system behavior are less meaningful.

Whether considered operational and system behavior, the horizontal interaction of system performer objects using commands and percepts as illustrated on SysML IBDs is the most dynamic description of the behavior. Since the signal interactions represent horizontal interaction, this IBD conveys different information than a more standard SysML IBD, which typically represents physical signal interactions between physical components. This use conveys similar information to a sequence diagram with the following distinctions:

(1) Though data exchange is not shown directly, it can be found by linking the interaction to the appropriate world state model. The data exchanged in many instances will have complex structure (e.g., cargo content), and not easily conveyed on a sequence diagram.

(2) Many variations are possible in the flow of commands and percepts through the system along with variations of the data that they include. A given command may include an entire mission assignment or one tactical object for update. Not clear how many sequence diagrams would be needed to convey this variation.

(3) ALOs are not typical OOAD software objects. They do not invoke a method and await a response, rather they interact with commands and percepts as independent actors that can operate concurrently. The behavior does not necessarily require a sequential thread.

Activity diagrams, FFBDs, and business process modeling are all ways to represent workflow. The workflow in this approach is received as an overall mission assignment where it is decomposed and allocated to various ALOs. Each ALO manages its mission state relative to that plan or assignment. The desired workflow is embedded segmented and embedded into the ALO models as part of their respective world state of interest.

Direct translation to component solutions. The system model is structured by interacting performer objects. For the most part, each of the performer objects can directly link one-to-one to a physical solution. These physical solutions could include a specific physical mechanical component, a controller board, a sensor, or software code. Some performer objects may also link to as an instance of a physical solution that can be aggregated into a physical solution. These include SC2A computation objects, ALOs and sensors. An ALO for example can be realized as software code, a neural net when combined with a computation instance, or by a physical instance of a human performing the ALO role. ALO and sensor physical realization need to be considered together. A given ALO realization as a human instance may also require an associated sensor to be realized by a human sensor of that same instance. World state models within the ALO can also be realized whole or in part by a physical data store, though the direct link may not be as obvious.

Component Assembly Independence. One of the advantages of software OOAD is that objects and their interactive behavior can be defined and analyzed to more granular levels of resolution and then assembled or aggregated into programs or software configuration items. The same principle applies to the system logic using this approach except the logic can be realized as software code, hardware, or human instances. Once selected, the objects can be aggregated up to higher level assemblies or configuration items. These assemblies together with the component selections impact how well the system achieves MOEs/MOPs as well as impact other system concerns, such as cost and mass properties. Alternative assemblies can be considered in conjunction with alternative physical component selections and the impacts addressed within a trade space assessment.

Design Science Method Verification and Validation Consideration The MCPS DM2 and associated foundational concepts applied to a MIGVS concept design is a design science method, specifically, a design science method for concept design of cyber-physical systems. The Validation Square (Seepersad et al. 2005) is proffered as a way to verify and validate research of a design method. As Seepersad et al. assert for a design method, "research validation is a process of building confidence in its usefulness with respect to a purpose." Table 5 shows the Validation Square, which is divided into four quadrants with a related set of criteria. Each quadrant is used roughly in order to build successive confidence or validation of the research method's purpose and usefulness. The numbered criteria in each quadrant corresponds to the same number in the dissertation research evidence column. As Seepersad et al. explain, the criteria of Quadrant 1) validates that the proposed design method is logically consistent. Quadrant 2) criteria validates that the example problems are appropriate to both illustrate and verify the design method. Quadrant 3) criteria is used to assess whether the design method produces useful outcomes. Finally, Quadrant 4) criteria is used to determine that the design method is useful beyond the specific case study and detailed example problems. The dissertation design method is validated based on the total set of criteria matched with the appropriate dissertation research evidence in Table 5.

Table 5. Validation Square Criteria with Dissertation Research Evidence

Validation Square Criteria	Dissertation Research Evidence
1) Theoretical (Domain-Independent) Structural Validation 1. Define design science requirements 2. Literature review to show advantages compared with other methods. 3. Characteristics for the domain are enumerated 4. Establish internal consistency	1. The requirements are captured in the problem statement and research questions and include: modeling multiple forms of behavior, technology independence, integrating operational and system behavior, and supporting component instantiation. 2. The literature review and research value indicate there is no current method that simultaneous achieves all the requirements identified for the MCPS class of system and problem, particularly the intelligence and human independent aspects of the problem. 3. Characteristics for the domain are enumerated within the foundational concepts (e.g., goal based state behavior, state change through interactions). 4. Both the concepts and the case study data utilize the same modeling language (i.e., SysML). Both structure and behavior considerations are captured in the DM2. The DM2 is consistent with the foundation concepts which in turn are consistent with the structure and behavior of the case study model. The structure and behavior of the case study model are internally consistent within and between each ALO as captured by the five integrated model diagrams.
2) Empirical (Domain-Specific) Structural Validation 1. Characteristics of example problems are similar to actual problems 2. Example problems cover all the needed characteristics 3. Examples produce data that can be used to compare with other methods	1. The performer object model encompasses a wide range of common MIGVS logic: material handling or special mission, mobility, IRSTA, TC3 and SC2A which includes mission command and control. Each performer object internally models intelligent behavior connected to a base CPS layer. The DM2 and the five diagram types capture needed characteristics: performer and context logical hierarchy and state, goal state mission assignments, interactions with other performer objects, internal ALO composition of behavior and world state models, and the ALO internal behavior that acts on world state and interacts with commands and percepts. 2. Each performer object covers a unique form of logic required by a MIGVS. Each performer object internally models multiple forms of behavior and is required: internally exhibits multiple forms of behavior. Integrated together the performer objects model all the forms of behavior and simultaneously achieve the requirements of the research questions. 3. The five types of diagrams produce data that can be compared to other methods when generalized. This includes missions, tasks, MOEs/MOPs, operator roles, physical mechanical and control based-functions, information management, and goal-based intelligence.
3) Empirical (Domain-Specific) Performance Validation 1. Do example outcomes meet requirements and characteristics 2. Does data support outcome conclusions	1. The five diagram types each capture key required characteristics as demonstrated. The five diagram types integrate with each other and other ALOs or sensor/controller objects to meet the requirements of the logical concept design overall. 2. Five performer objects with 34 ALOs that each model a part of the required intelligent behavior, provide depth and breadth of modeling sufficient to inform concept design and meet the stated research requirements. The logic defined in a component framework can support a trade space exploration that can include a trade relative to the number human operators.

Validation Square Criteria	Dissertation Research Evidence
4) Theoretical (Domain-Independent) Performance Validation 1. Domains with the precise characteristics 2. Examples of more general class 3. What characteristics are applicable	1. Any MIGVS has nearly the exact same characteristics as the PLS. The only exception would be lethality which has similar characteristics and could be modeled in a similar pattern as material handling and mobility. 2. A more general class is MCPS or any CPS that has hierarchical control logic with a large number of states, including goal states. 3. The characteristics applicable for another domain for relevant use are: hierarchical control logic with a base CPS layer, a dynamic or otherwise state intensive system and external environment, and intelligence that is goal directed and context aware.

V. CONCLUSIONS AND RECOMMENDATIONS

This dissertation contributes a modeling framework capable of modeling the complete behavior logic of cyber-physical systems as a set of interacting and abstract component-based performer objects: agents, controllers and sensors. An advantage of the modeling framework is all types of system objects are modeled using the same modeling language constructs. As a result, behaviors that would be allocated to physical hardware, software and/or human elements are all modeled, analyzed, and treated the same. The modeling framework's agent and component abstractions are technology and implementation independent yet meaningful enough to capture key performance attributes, attribute dependencies, and the key behavior logic required of the system. This dissertation demonstrates how the modeling framework performs the initial MIGVS concept design of system behavior. The framework's interacting components can be instantiated as hardware, software and/or human roles and the impact on that behavior assessed as part of a trade space exploration during the concept design phase. Current modeling methods are unable to treat hardware, software, and human elements on an equal footing using a single or integrated modeling approach.

When the initial behavior or logical concept design is integrated with a 3D CAD concept design of the physical architecture and it's mass properties, a trade space exploration is enabled that increases the number and type of components and related capabilities that can be considered and therefore increases the number of possible combinatorial system solutions. The cyber components (e.g., electronics, sensors, software), and their contribution to operational effectiveness and system capability, will have a visibility on par with the traditional physical-mechanical components and their behaviors. Furthermore, the interdependencies between cyber and physical-mechanical components can be established. The cyber components can be assembled and integrated into a CAD physical architecture at an equivalent level of abstraction. The final concept design can reflect both hardware and software configuration items as well as any crew configuration. Logical design portion of this final concept design may be used just for

analysis or it can become part of the system architecture or technical baseline and retained through the life cycle.

The MCPS concept data meta-model (DM2) and associated concepts provide the necessary perspective and framework to initially conceptualize a MIGVS. The intelligence logic of a MIGVS is composed of a hierarchy of agent logical objects (ALOs) operating on a base cyber-physical layer. These ALOs are state intensive; that is, they have a state space with many variables that change as ALOs interact with each other and with context objects. At a certain level of abstraction, this intelligent logical hierarchy is independent of how it is physically implemented and assembled, to include human or technology realization. The interacting ALOs model the system intelligent behavior at that level of abstraction yet reflect component granularity below physical assembly. As ALOs are realized by particular physical implementations and aggregated into particular assemblies, the performance attributes of the ALOs and their dynamic interactions become physically constrained. Each physical implementation and assembly approach considered will change the performance. Performance changes can at least be descriptively assessed relative to physical component and assembly selections within a trade space exploration.

An ALO's behavior is not determined by its previous state and a current input (i.e., it is not simply a state intensive version of a finite state machine). ALO behavior is determined by the relation of the current state as passed to it from a subordinate agent and its goal state as commanded by a superior agent. The goal state requires specification for ALO potential realization by machine versus human and to facilitate trade space evaluation (i.e., how well does the machine perform relative to the human?). The collective ALO goal state specifications equate to the system's operational measures of effectiveness or performance. An effective specification of a goal requires state and time target measures as well as tolerances for both. It also requires a way to measure loss of value from the target. The goal specification coupled with the component granularity of the ALOs results in a more precise and comprehensive definition of operation behavior while preserving solution independence.

The system concept is not fully defined without including data concepts that reflect goal state and state variables. State as reflected in data, information, or knowledge can be categorized into three types (Evans et al. 2002): parametric as used by simple or mathematical functions including control systems, complex structures as used in dynamic context sensing and information processing, and symbolic as used for high level reasoning. As Evans et al. explain, these complex data structures include spatial geometry, maps, images, and can also include arrays and lists. The data concepts when realized by machine must be accompanied by the necessary sensor concepts to acquire the data. The collective intelligence of a MIGVS requires all three data types to sufficiently understand its goals and dynamic world state and distinguishes it from typical artificial intelligence approaches using only symbolic knowledge or data. The MIGVS concept must include and integrate data concepts along with the physical-mechanical concepts and cyber component concepts.

The ALO hierarchy is a cyber or computational control hierarchy that should enable a reusable structure of application logic or software. Any MIGVS can be arranged the same pattern as used in the case study. A mission agent is at the top level of the hierarchy and controls task agents at the next level. Each task agent interacts with an intelligent and detection agent layer, and they in turn control sensors and controllers. MIGVS as a domain has similar logic across systems. For example, a PLS participating in a convoy mission is nearly identical to a combat system participating in a tactical road march. Maintaining local situational awareness is a common solider task within different systems that can be realized by an IRSTA ALO. The application logical hierarchy does not change no matter its physical realization. For example, the intelligence could be realized: by a single computational assembly with fan out connections to the sensors and controls, by a distributed one for one match of ALO and computational assembly, by a one for one match of ALO with “human assembly” resulting in dozens of crew members, or by a myriad of crew member and computation assembly combinations. To the extent that ALOs are realized in software, the hierarchical application logic can stay the same whether it is realized in a single program on a single computer or distributed across

many. The performance can certainly be impacted as the interaction between ALOs take different paths thought the vertical logic.

From a MIGVS domain standpoint, the application logic reusability should also pay off in terms of generating system models with associated operational architectures and hardware components. Standard performer and context objects structures can add value in terms of generating system models where the operational behavior logic is common. Where the operational performance is nearly common, then reuse of physical implementations whether software, hardware or human, should also be a distinct possibility. Reusable application logic together with components that conform to that logic would enable portfolio management using rapid generation of well-specified and high fidelity operational architectures supported by system OOAD to define feasible systems. Advancements in capability can be specified in operational terms and system feasibility assessed relative to existing and/or new technology.

1. Limitations

The logical modeling framework is limited by model complexity, unfamiliar description, and lack of a supporting analytical framework. The relative higher fidelity of the logical model as compared to more typical systems engineering modeling approaches results in complexity that will require more time to generate. The model also has many crosscutting relationships that must be synchronized and cannot be practically generated without modeling language support. The native presentation is not as familiar or intuitive as other methods such as business process modeling or functional flow modeling and may require presentation translation, particularly for non-technical stakeholders. As a new modeling method it does not have a ready-made analytical support tool or framework. Not clear how discrete event simulation would be used since a given event only results in a behavior when the current state change is significant relative to a goal state. Agent based simulation would seem an obvious support tool, but these agents are higher fidelity and act on complex data structures similar to an actual system. The logic required to make a given ALO executable would approach that of a software prototype.

The patterned nature and solution independent reusability of the approach may mitigate some of these limitations.

The logical model development approach is limited by the lack of needed source material and reliance on existing doctrine and tactics. The approach leverages the use of design reference mission and "design reference system." Necessary source information may be spread out over many training procedures and other doctrine, often not quantified sufficiently, and difficult to determine whether critical information might have been missed. Current methods for specifying operational requirements and architectures do not provide the necessary information for this method. Establishing an authoritative source of the necessary requirements will take additional time. Finally, many of the advantages of this method may be neutralized for new systems using significantly new doctrine and tactics, particularly where the capability exceeds that of previous human operators, for example some sort of precision maneuver. In this case, perhaps a comparative point of departure could be used relative to the design references. Also, assuming new tactics are not invented for each new system, this method would still apply to future upgrades and similar systems.

2. Future Work

Three significant areas of future work are recommended. The first is centered around the System 4+1 model shown in Figure 53 and consists of taking the initial concept design as defined here or similarly, and define a final concept design and initiate a development. The concept design would require concurrent development, iteration, and the appropriate integration relationships between the following:

(1) Logical Model—physical component selection of the performer objects to include human roles or instances. This selection would include which ALOs would be realized as software. Test the handoff the object models for usefulness and problem understanding for software developers. Retain the final logical model as a decomposition bill of material (DBOM). Define user interface solutions as needed. Update the domain performer model as required.

(2) Physical Model—harmonize DBOM component selections with concept CAD EBOM selections. Adjust DBOM solutions as required to meet integration and mass property constraints.

(3) Execution Model—assess ALO defined software the computational architecture selection. Assess performance and concurrency impacts.

(4) Deployment Model—complete final concept deployment of the physical architecture. Include software deployed to distribution computation components and embedded networks.

The second recommended future research is to fully elaborate all types of vertical logic. This effort succeeded in identifying SC2A tasks and trajectories and isolating the vertical from the horizontal logic. Only an example was shown as to how ALOs connect to computation and signal objects. All ALOs can be similarly connected to computational and signal objects and the computational objects decomposed into its constituent "stack" objects. The computational "stack" objects may be different for different types of ALOs, particularly for detection agents that need to process complex data, such image processing for recognition. Full elaboration of the computational logic would have to be done within the framework of an execution model. The execution model may be a fully integrated vertical and horizontal logic model with physical component selection and attribution. A full examination of the vertical logic would require system level interrupt handling to handle exceptions. Exceptions are likely to be the stressing case on sensor and computational performance. Also, a full system model requires power distribution and structural support. Both of these are a type of vertical logic that can impact horizontal logical performance. A structural component for example must reliably execute its purpose. If not, there is likely an impact to the performer object(s) that requires that support to fulfill its purpose.

The third recommended future research is to take advantage of the performer object structure and define time dependent mission reliability for the system. ALOs will have a reliability that is only as good as the information it is utilizing. Information reliability can be defined to be one minus the information uncertainty. This information has both a source and a time dependency. The source dependency can be a matter of trust, as provided from an external source, or a matter of performance as in a detection agent

and sensor pair with Type I and II errors. It would be desirable to be able to tune these errors dependent on mission rules and conditions. Information, no matter the source, will degrade with time. A direct corollary can be drawn to hardware objects. Hardware will have source as delivered from the factory reliability, and a time dependent aspect that degrades with use and storage. Together these measures could be used to define a mission reliability or expected value that could vary in real time as the mission executes.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. PLS SYSML MODEL DIAGRAMS

These diagrams augment the diagrams shown previously. The MA diagrams were covered comprehensively previously so what follows is for the five performer logic. Infinite number of views are possible, so the views follow the same pattern as presented for the mission agent and material handling. They serve the same purpose as previously explained and are shown here for model completeness.

B. MATERIAL HANDLING

Material Handling (MH) was extensively discussed previously to illustrate the overall model and approach.

1. MH Assigned Mission

In addition to the tasks and trajectories previously discussed, MH has the following trajectory elaboration.

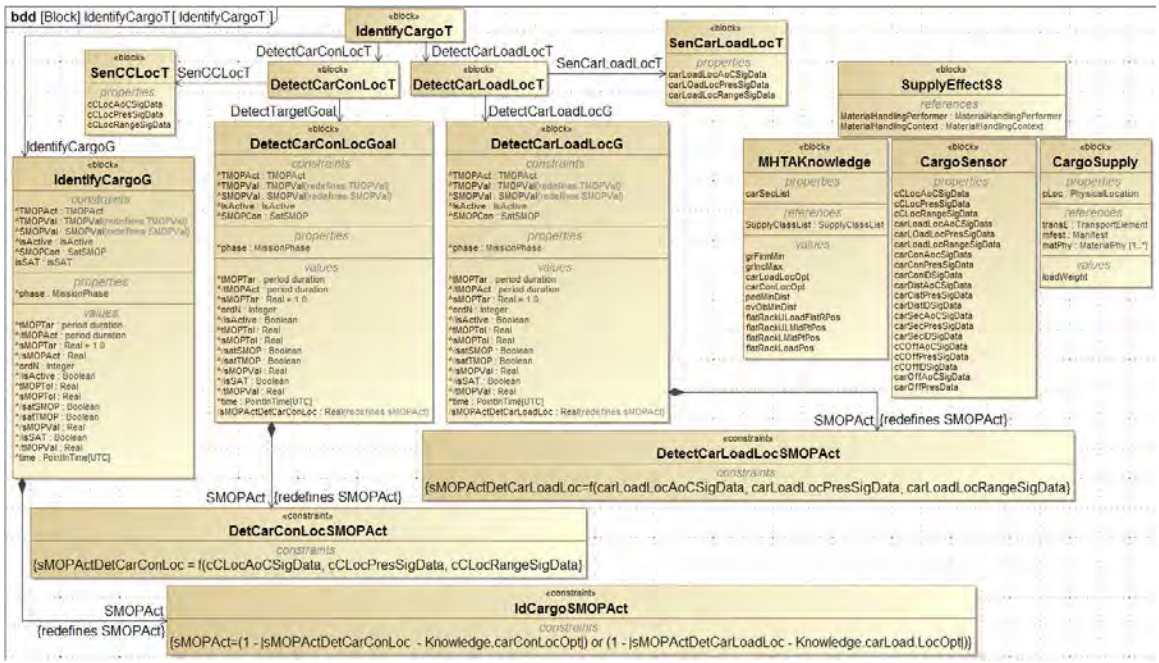

Figure 95. MH Identify Cargo Desired Trajectory

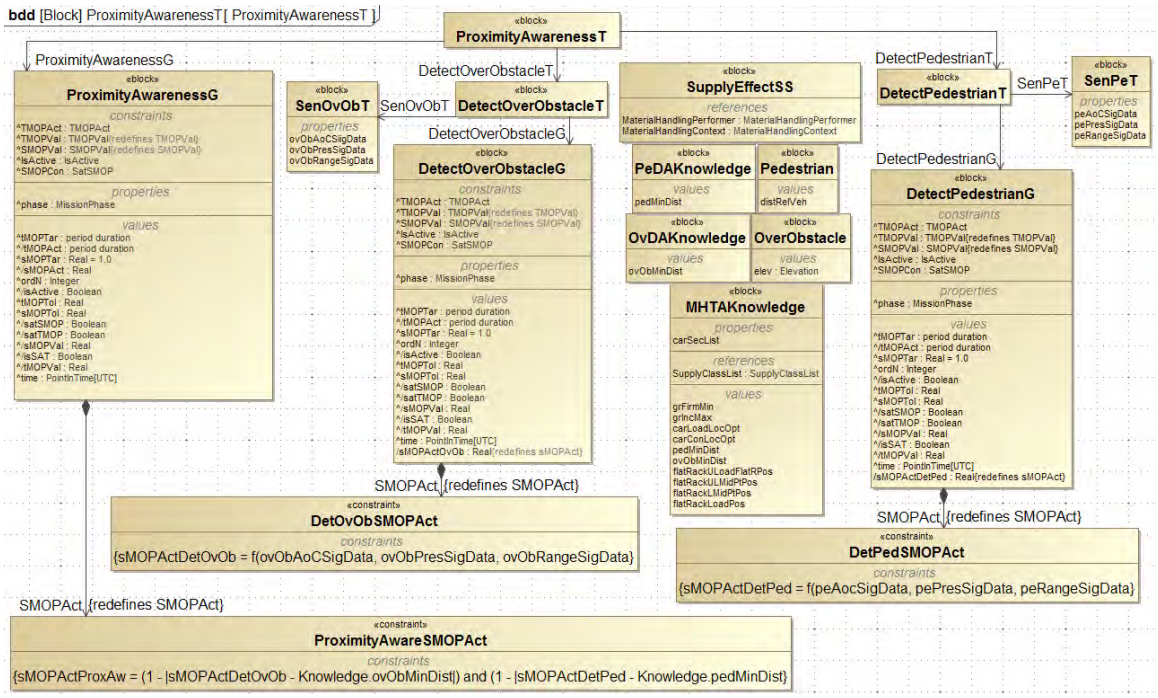

Figure 96. MH Proximity Awareness Desired Trajectory

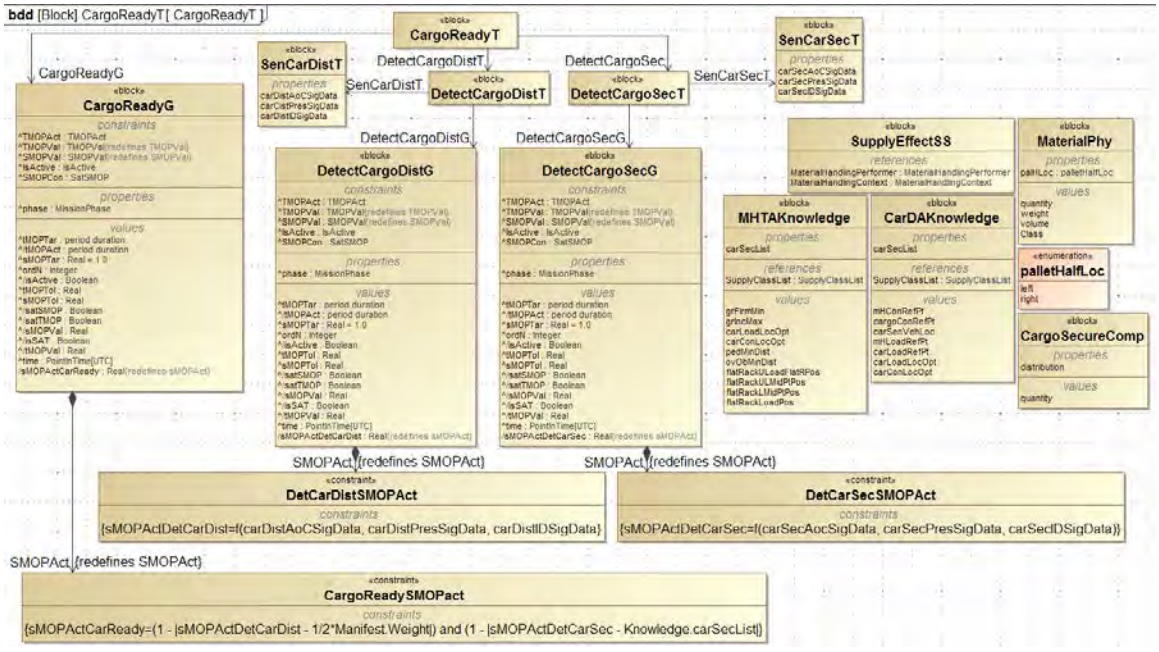

Figure 97. MH Cargo Ready Desired Trajectory

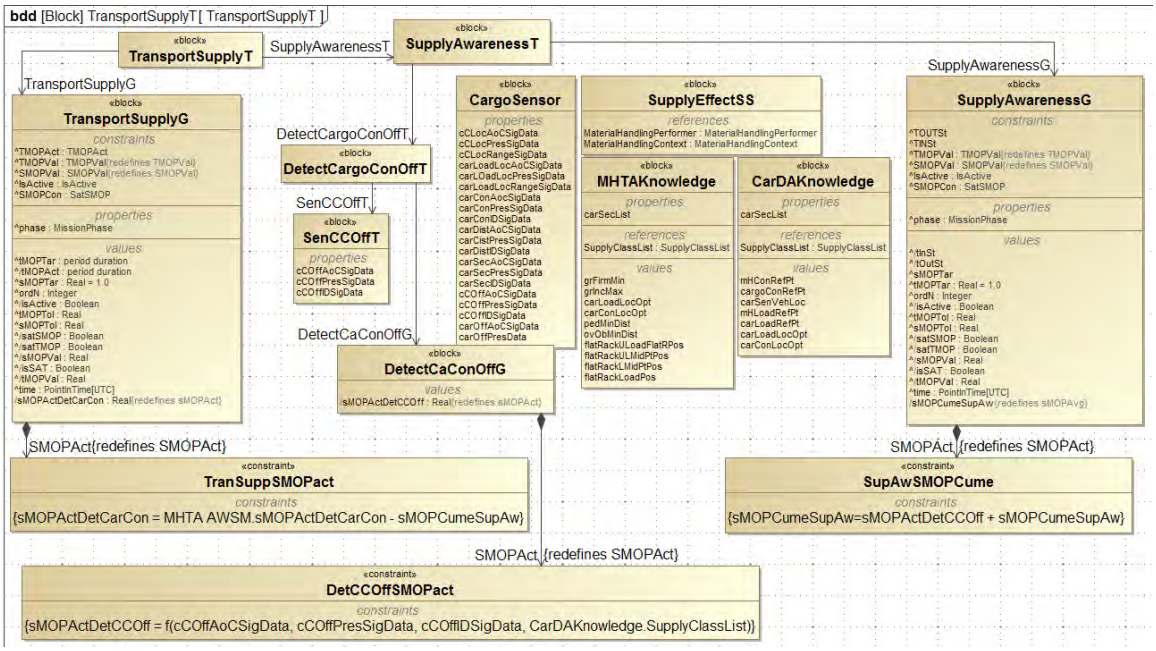

Figure 98. MH Supply Awareness Desired Trajectory

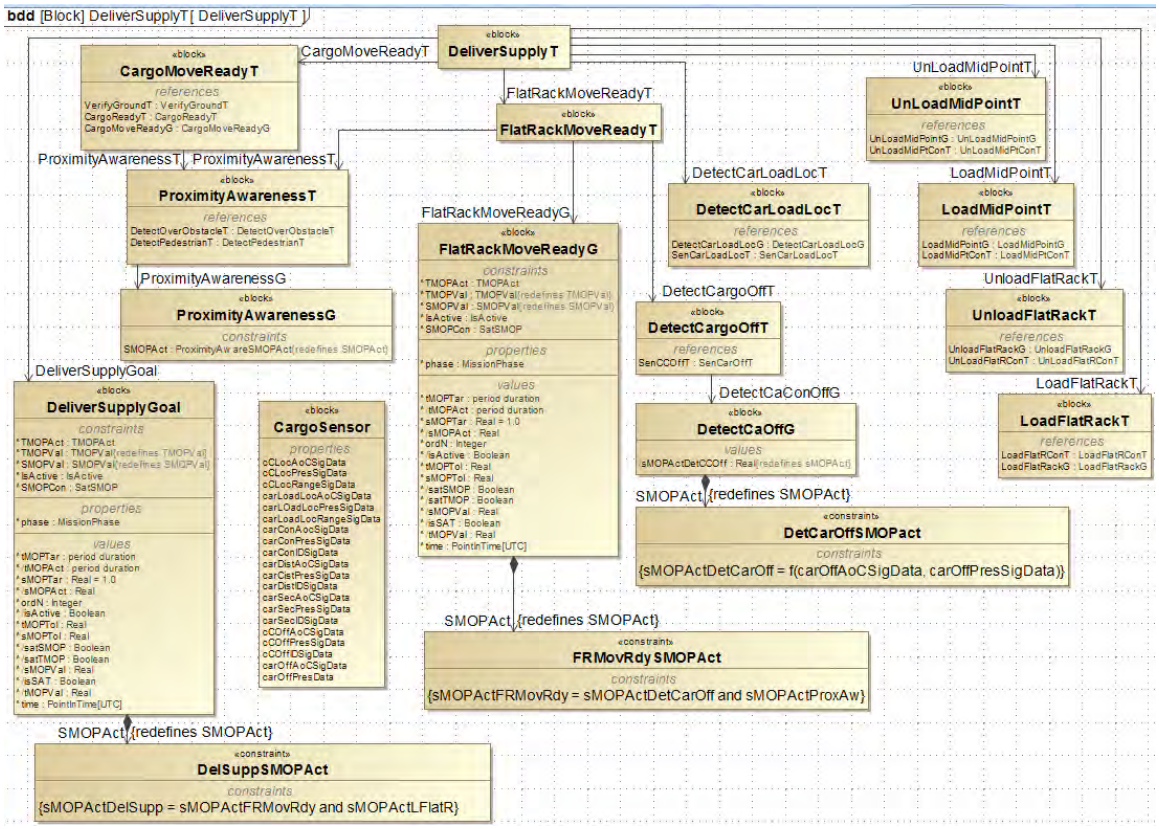

Figure 99. MH Deliver Supply Desired Trajectory

2. MH Logical Object Hierarchy

No additional material handling objects then already presented.

3. MH Interactions

There are no more addition MH IBDs then already presented.

4. MH Agent Internal Composition

The cargo detection agent was previously presented. The following are the other three detection agents for material handling.

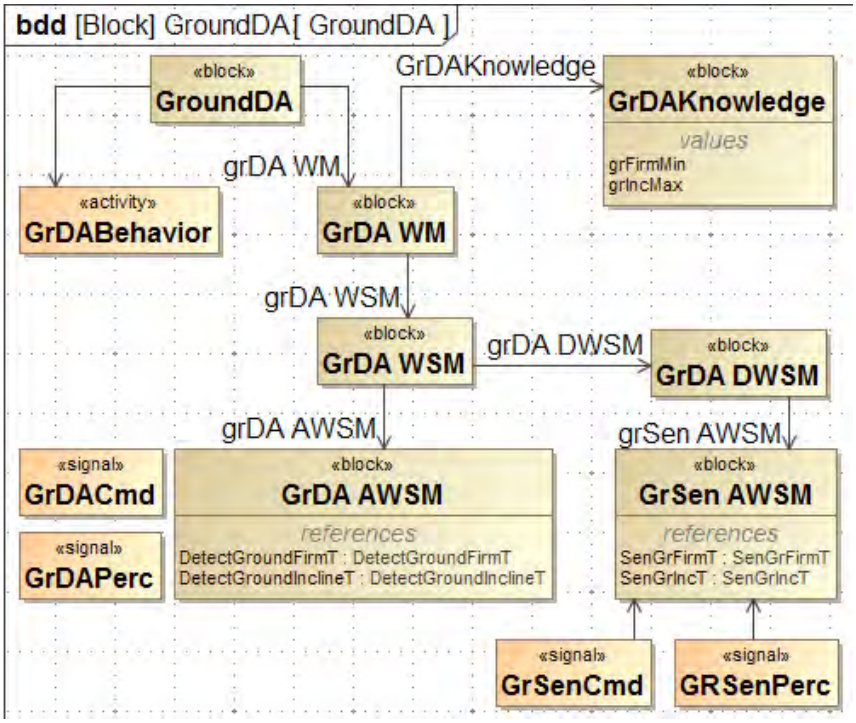

Figure 100. MH Ground Detection Agent

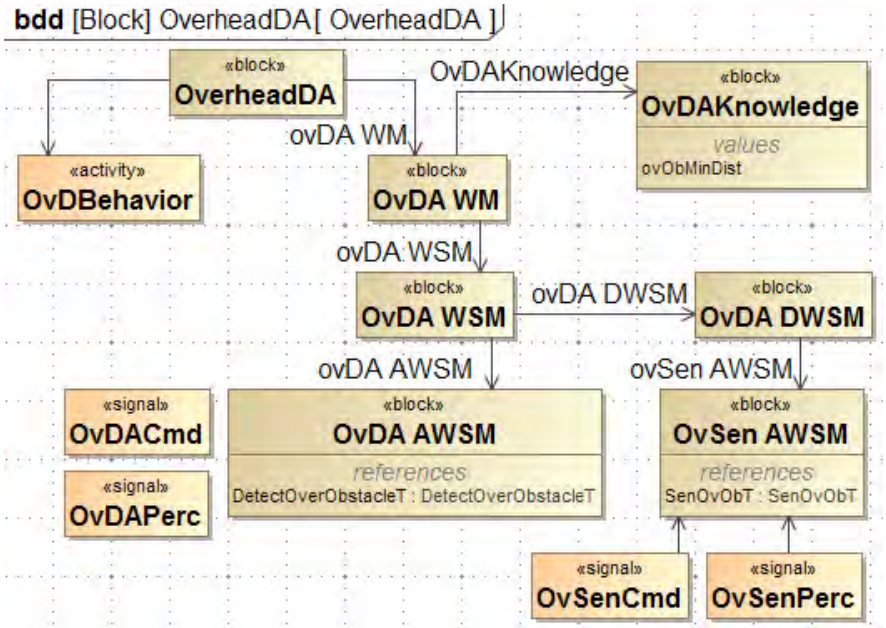

Figure 101. MH Overhead Detection Agent

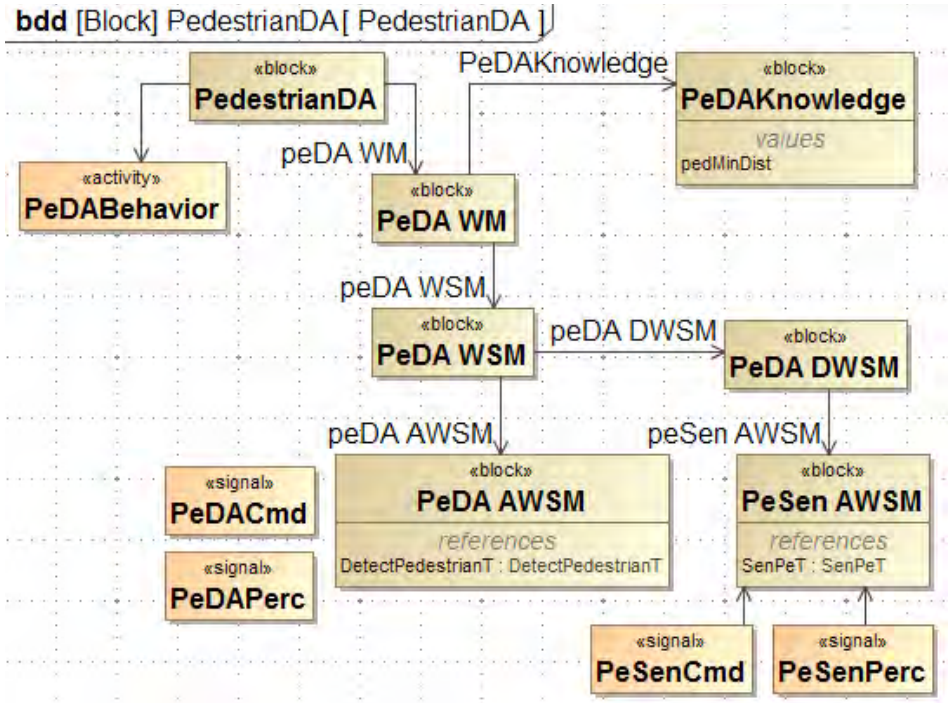

Figure 102. MH Pedestrian Detection Agent

5. MH Detection Agent Behavior

In addition to the Cargo Detection Agent behavior, MH has the following detection agent behavior:

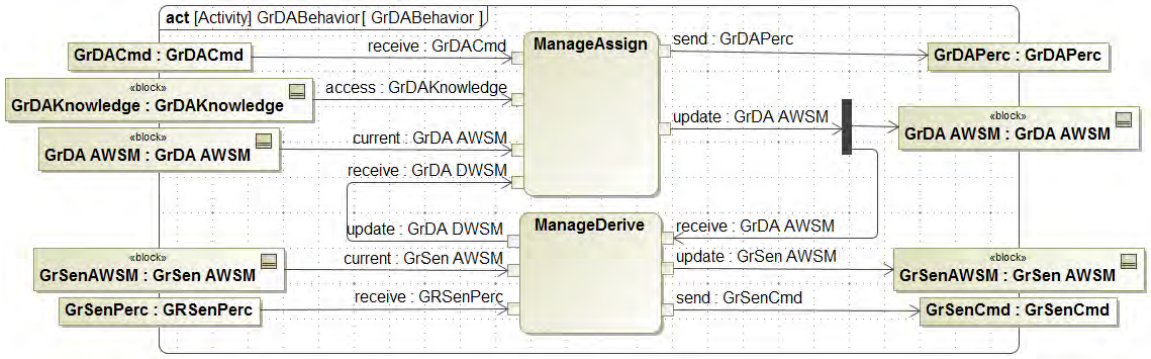

Figure 103. MH Ground Detection Agent Behavior

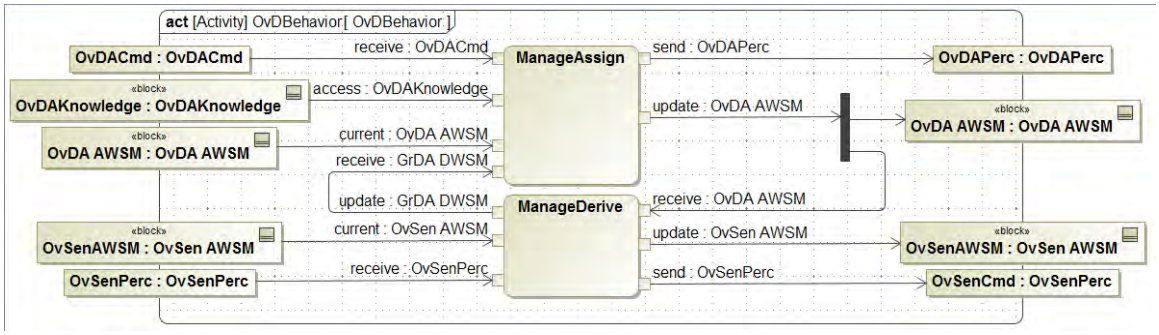

Figure 104. MH Overhead Detection Agent Behavior

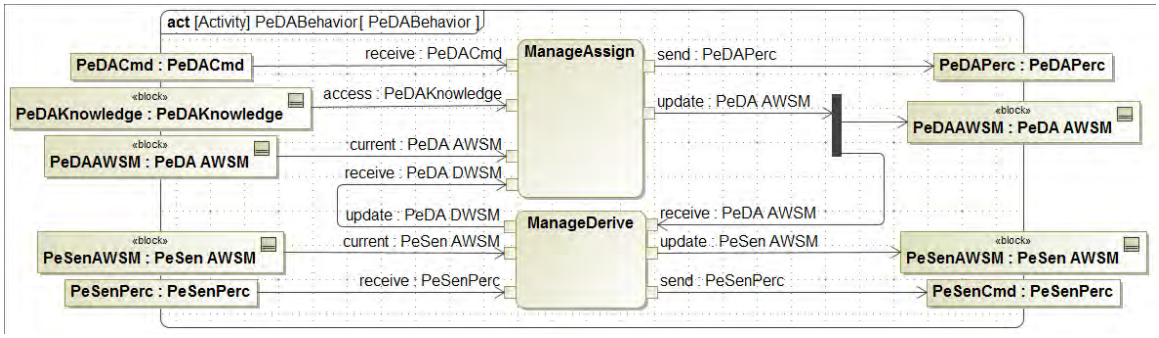

Figure 105. MH Pedestrian Detection Agent

C. TACTICAL COMMAND CONTROL AND COMMUNICATIONS

Tactical Command Control and Communications (TC3) was only briefly discussed previously. The full complement of diagrams is presented following.

1. TC3 Assigned Mission

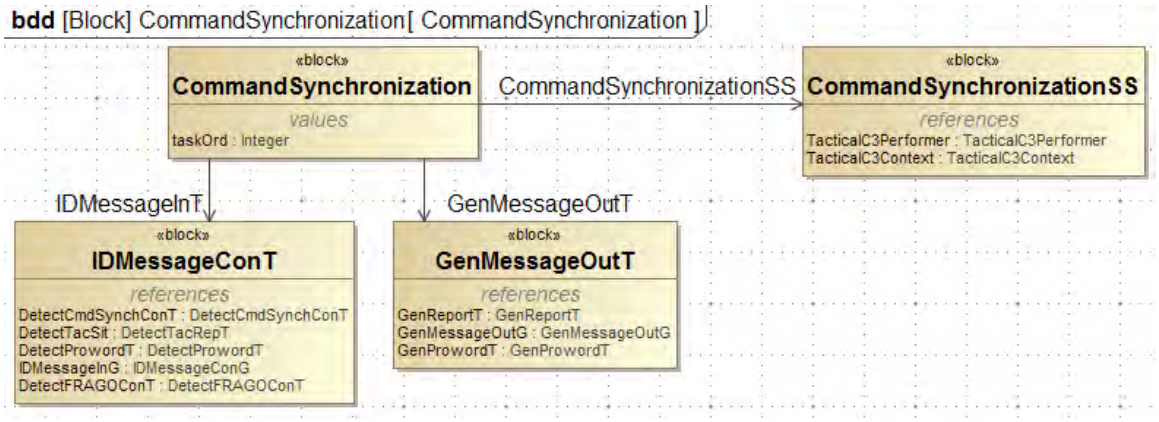

Figure 106. TC3 Command Synchronization Task

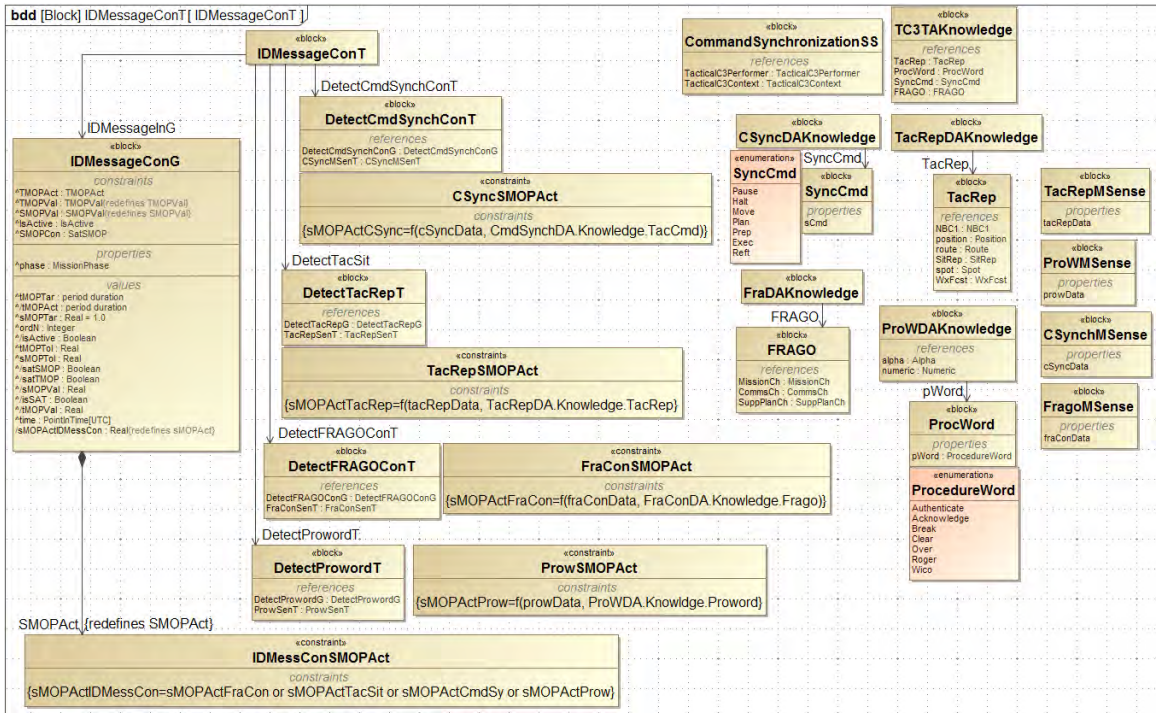

Figure 107. TC3 Identify Message Content Desired Trajectory

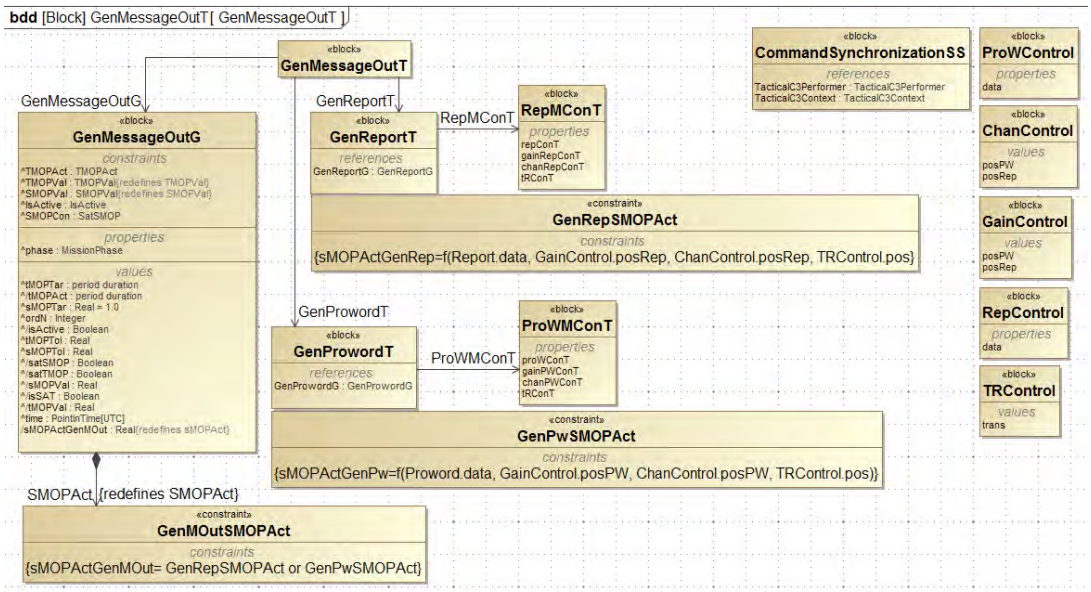

Figure 108. TC3 Generate Message Out Trajectory

2. TC3 Logical Object Hierarchy

None of the TC3 performer objects directly senses or effects the context. Its state is determined from interpretation of the data over the network and comparison to stored knowledge or preset knowledge (e.g., call signs). As such, there is not context logical object hierarchy.

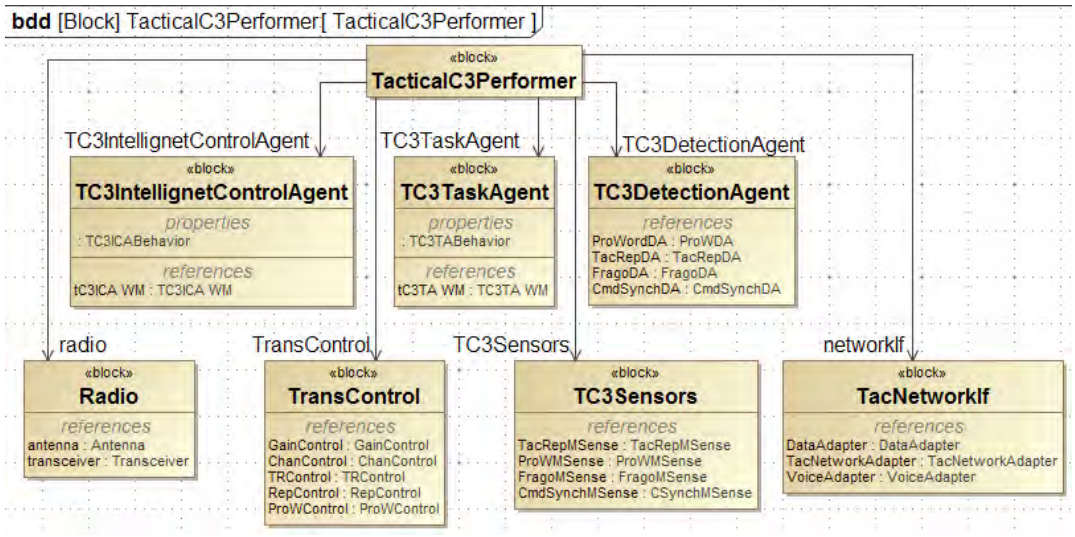

Figure 109. TC3 Performer Logical Object

3. TC3 Interactions

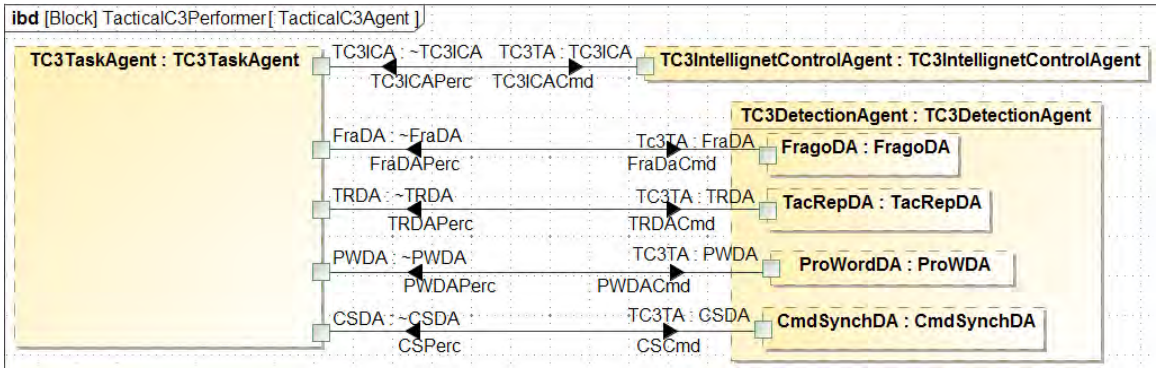

Figure 110. TC3TA and TC3ICA/Detection Agent Interactions

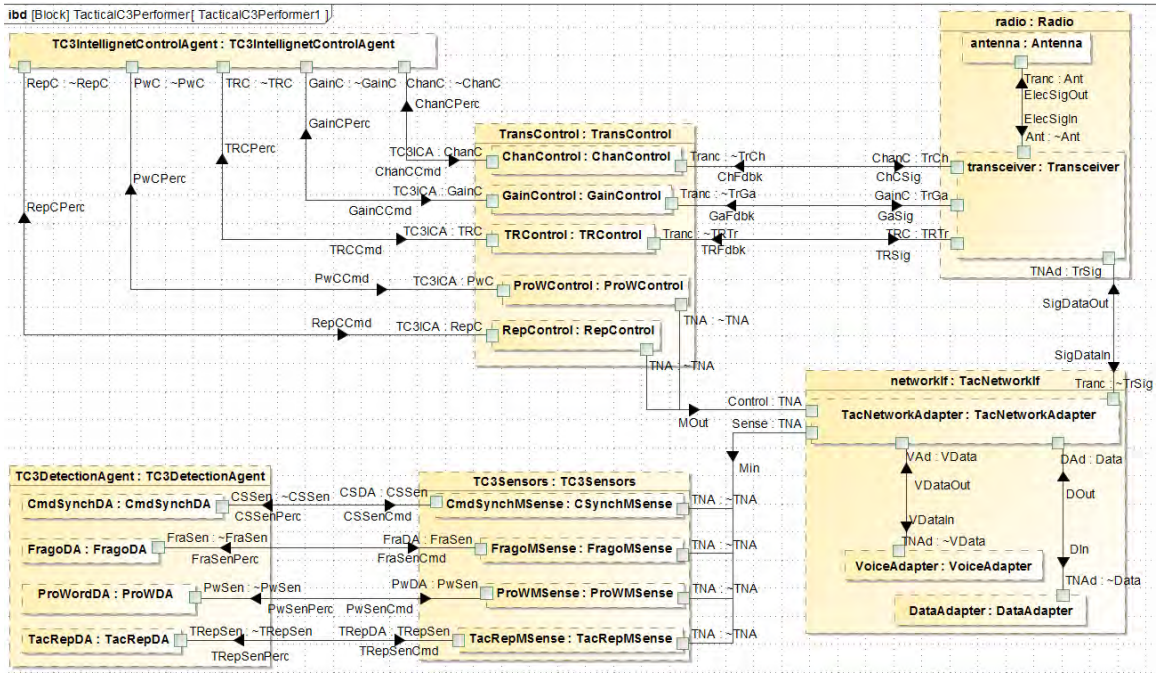

Figure 111. TC3ICA and TC3 Detection Agent with Control and Sensing Interactions

4. TC3 Agent Internal Composition

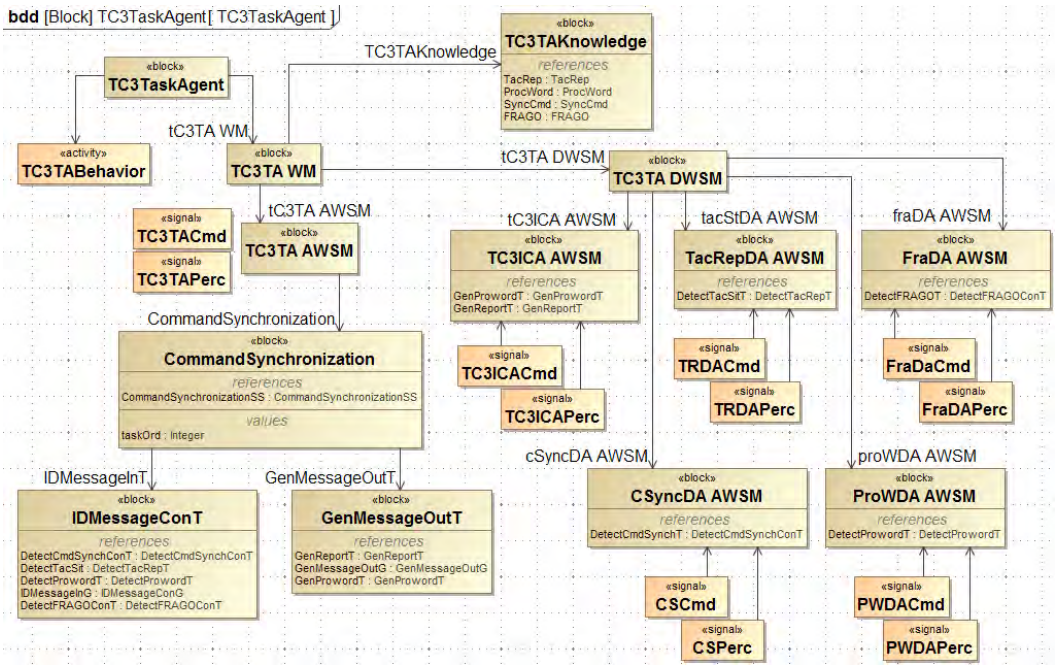

Figure 112. TC3 Task Agent Internal Composition

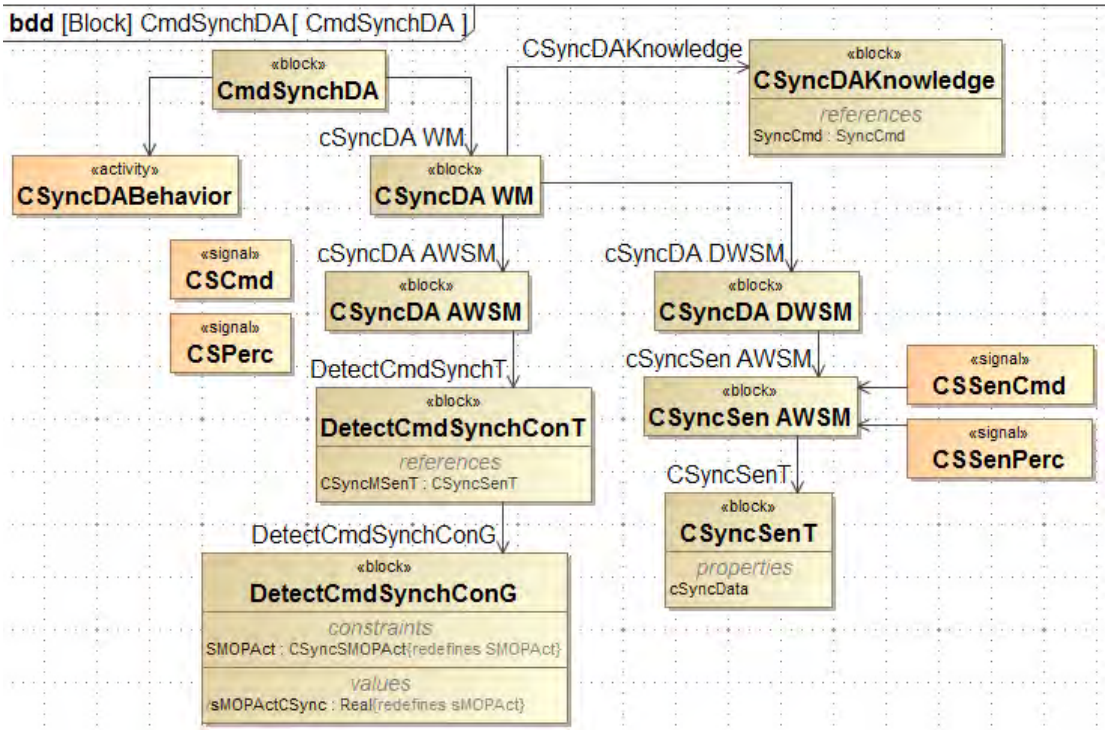

Figure 113. TC3 Command Synchronization DA Internal Composition

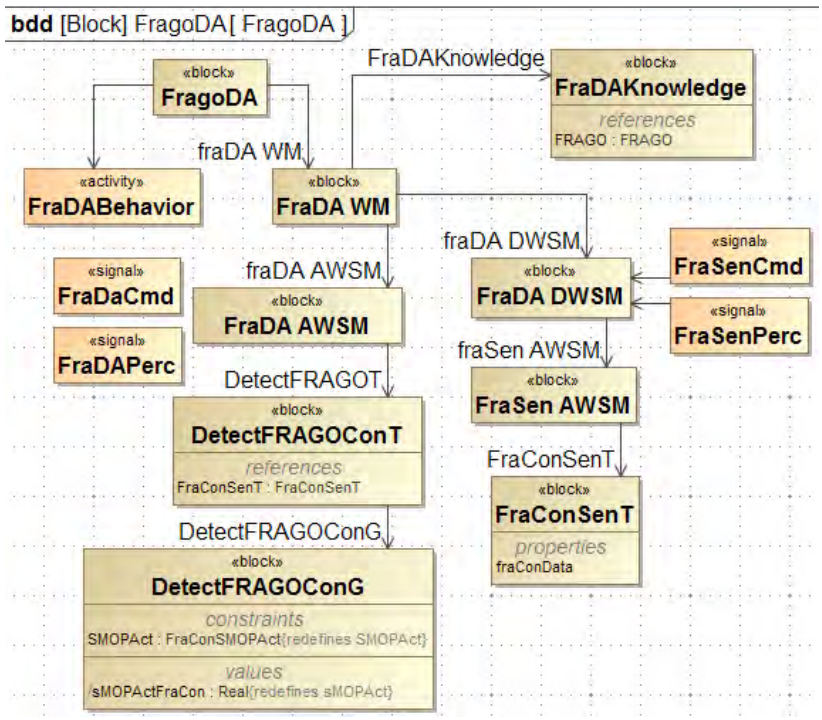

Figure 114. TC3 Fragmentary Order DA Internal Composition

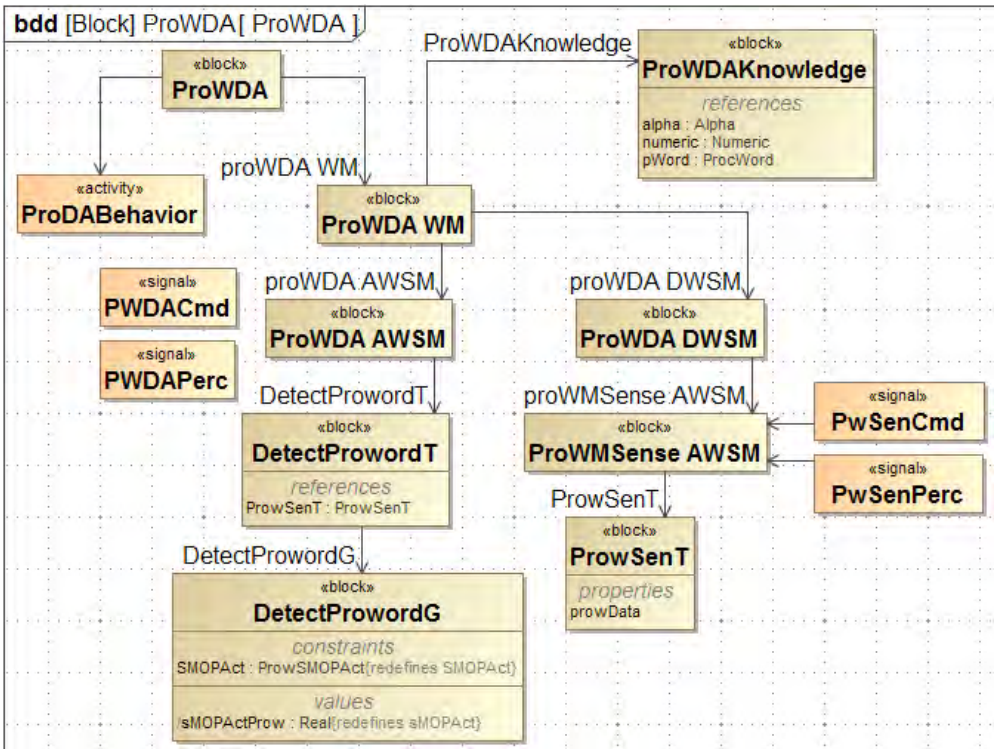

Figure 115. TC3 Pro Word DA Internal Composition

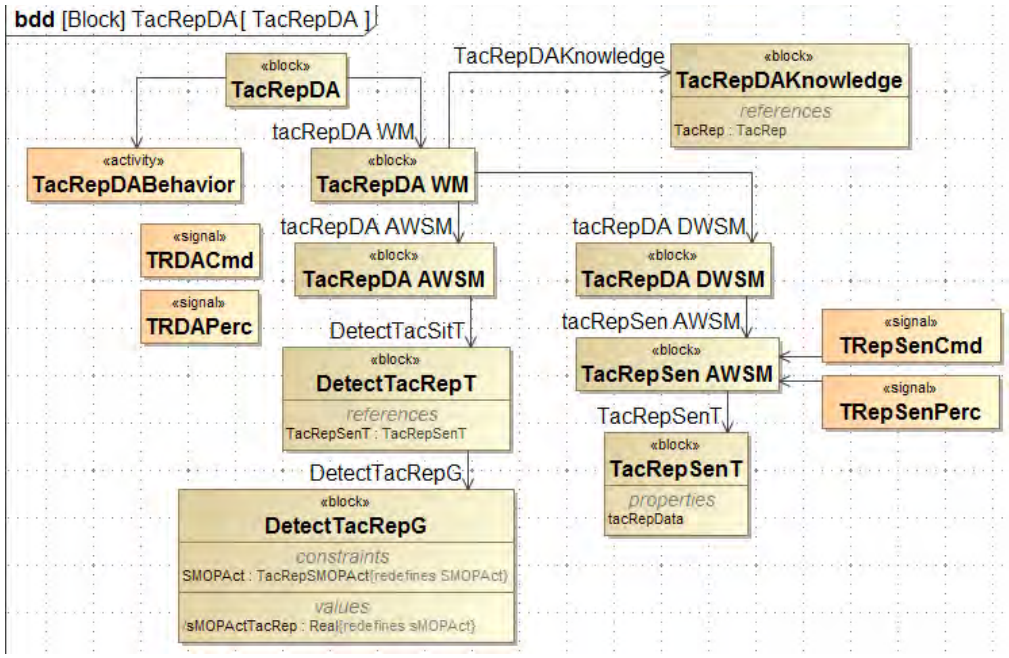

Figure 116. TC3 Tactical Report DA Internal Composition

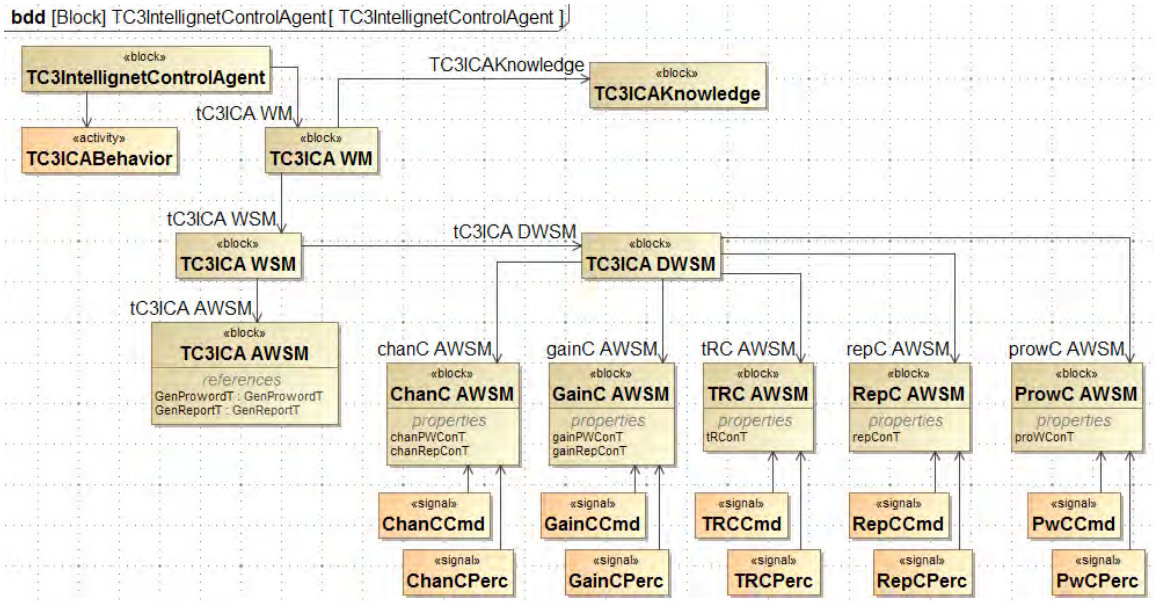

Figure 117. TC3ICA Internal Composition

5. TC3 Agent Behavior

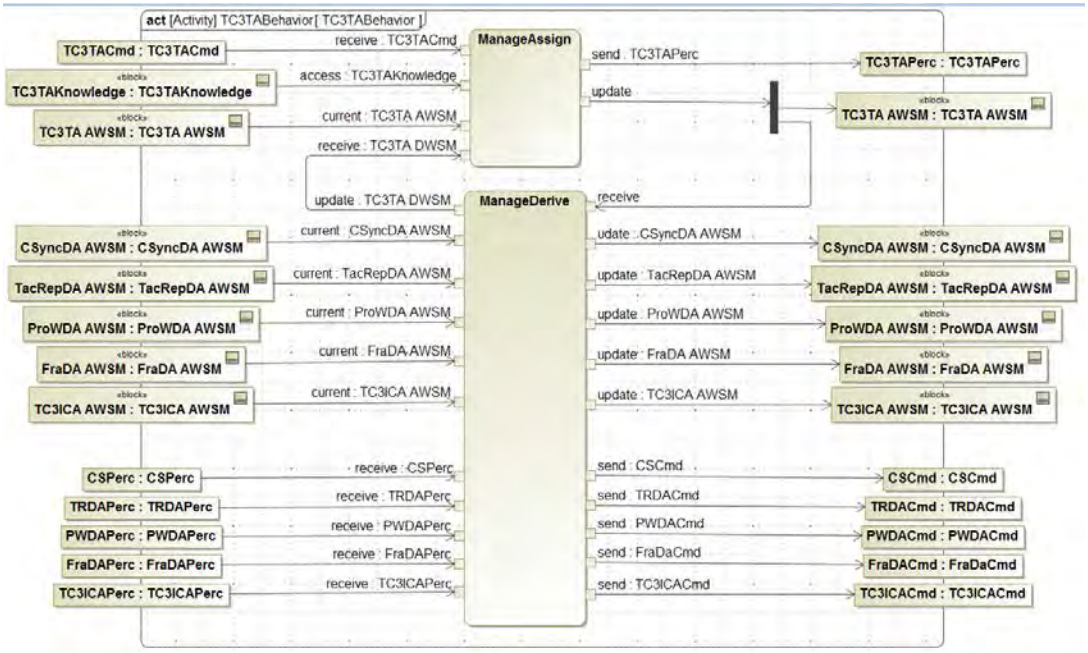

Figure 118. TC3 Task Agent Behavior

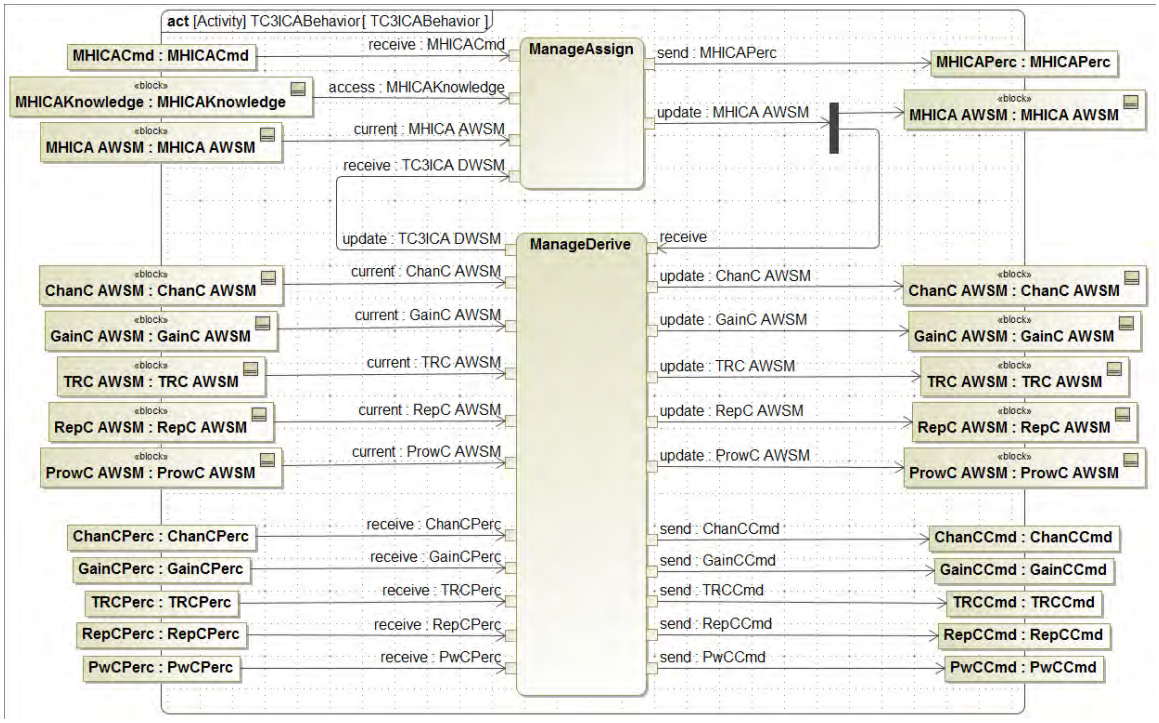

Figure 119. TC3 Intelligent Control Agent Behavior

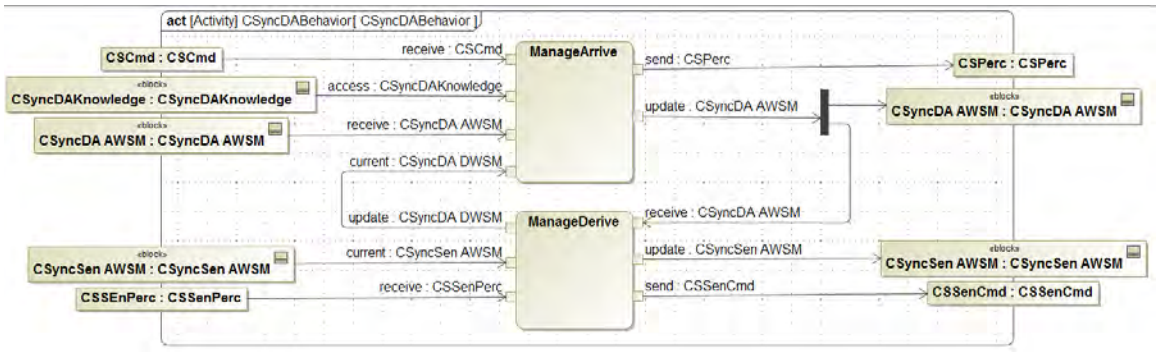

Figure 120. TC3 Command Synchronization DA Behavior

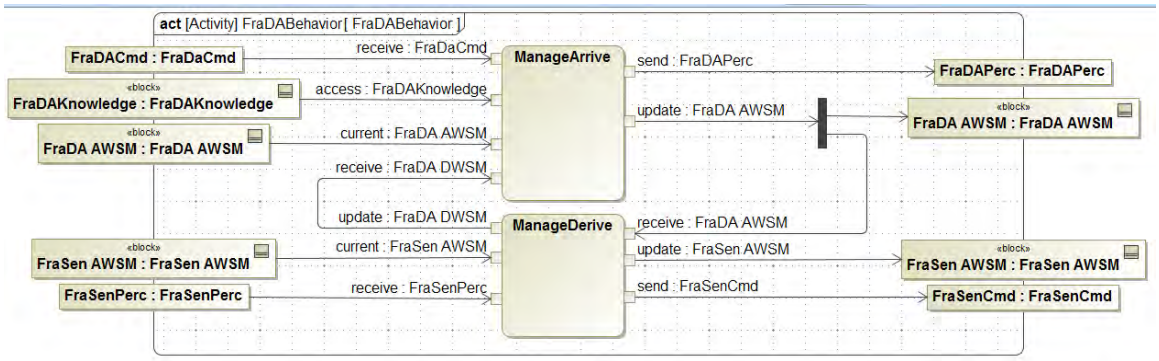

Figure 121. TC3 Fragmentary Order DA Behavior

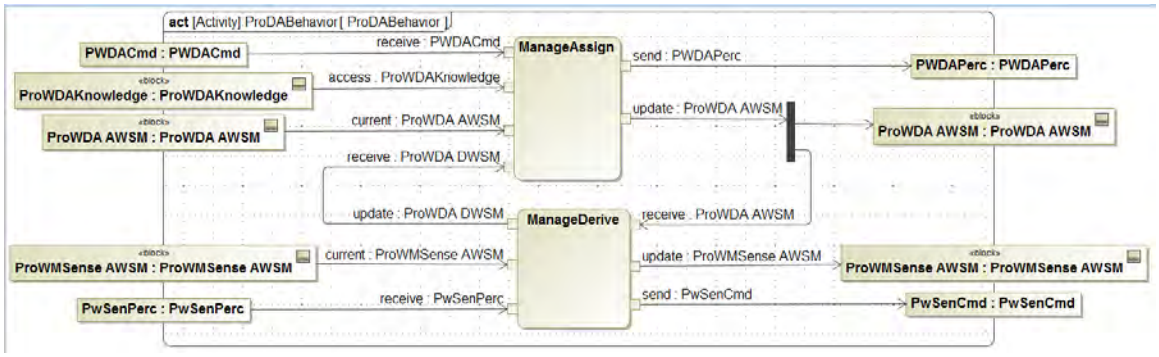

Figure 122. TC3 Pro Word DA Behavior

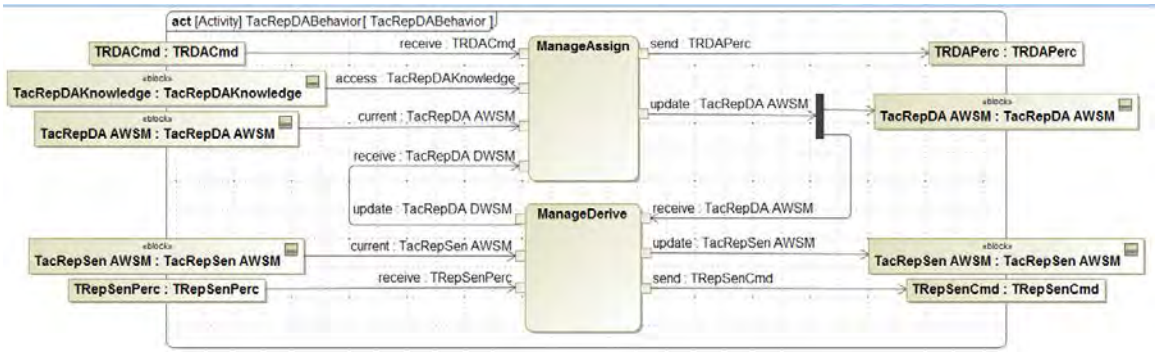

Figure 123. TC3 Tactical Report DA Behavior

D. INTELLIGENCE RECONNAISSANCE SURVEILLANCE AND TARGET ACQUISITION

Intelligence Reconnaissance Surveillance and Target Acquisition (ISRSTA) was only briefly discussed previously. The full complement of diagrams is presented following.

1. IRSTA Assigned Mission

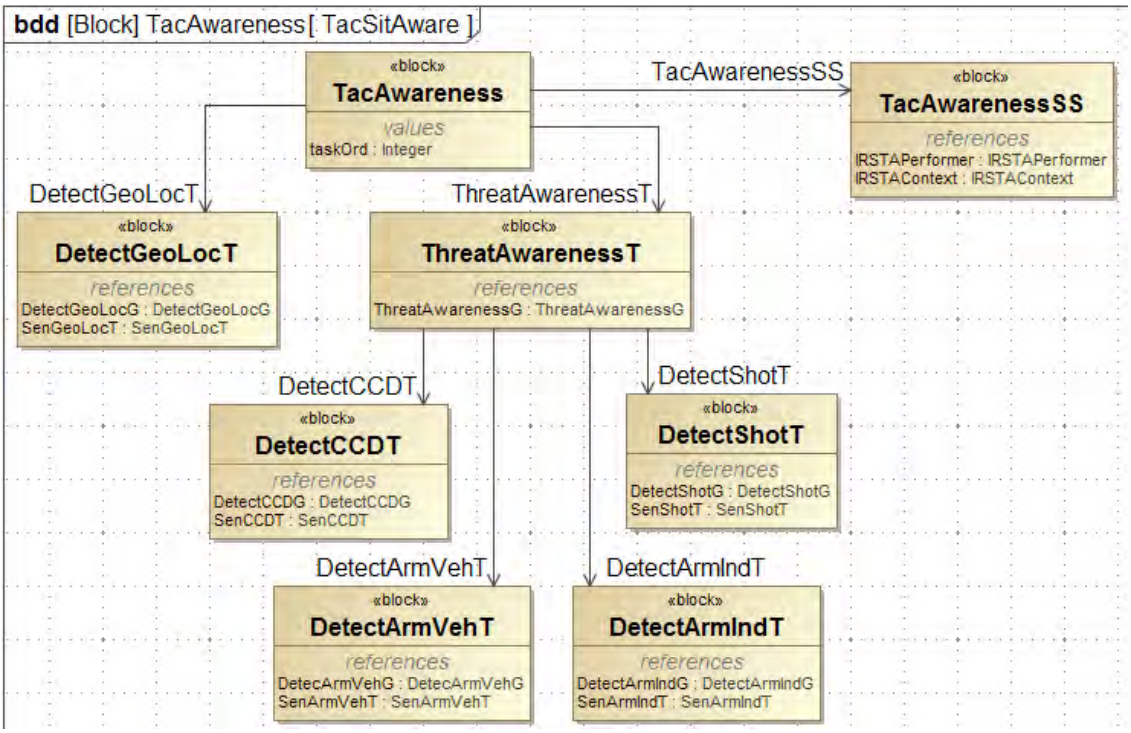

Figure 124. IRSTA Tactical Awareness Task

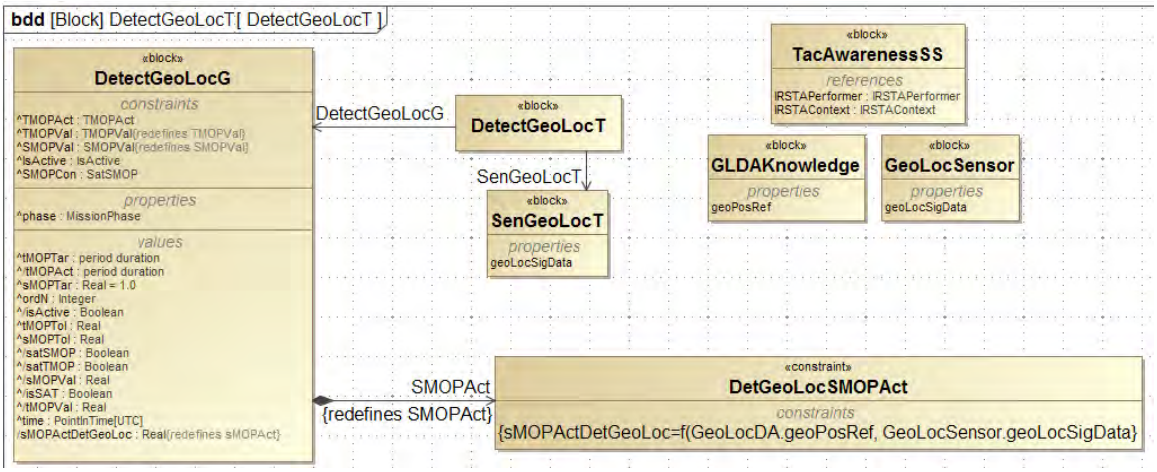

Figure 125. IRSTA Detect Geo Location Desired Trajectory

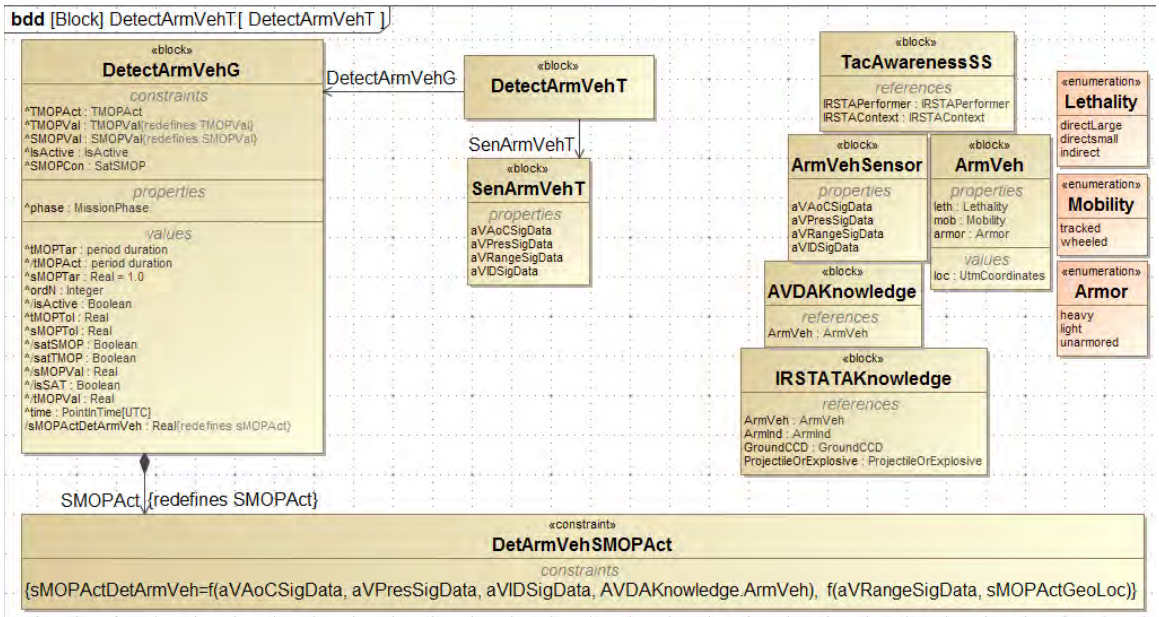

Figure 126. IRSTA Detect Armored Vehicle (and its geolocation) Desired Trajectory

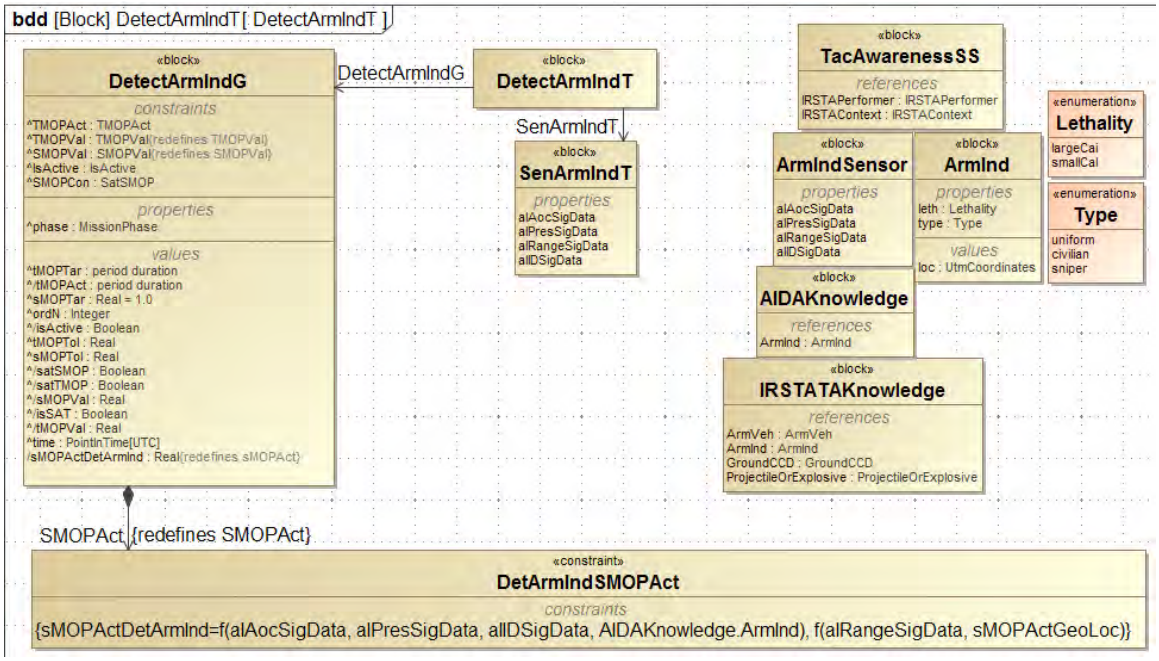

Figure 127. IRSTA Detect Armed Individual Desired Trajectory

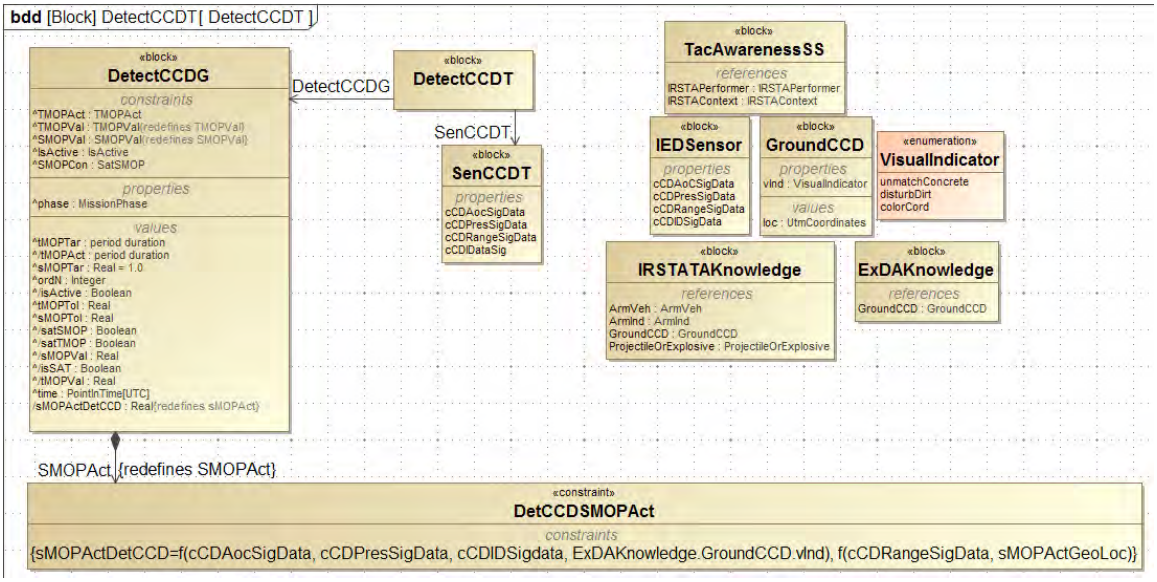

Figure 128. IRSTA Detect "IED" Desired Trajectory

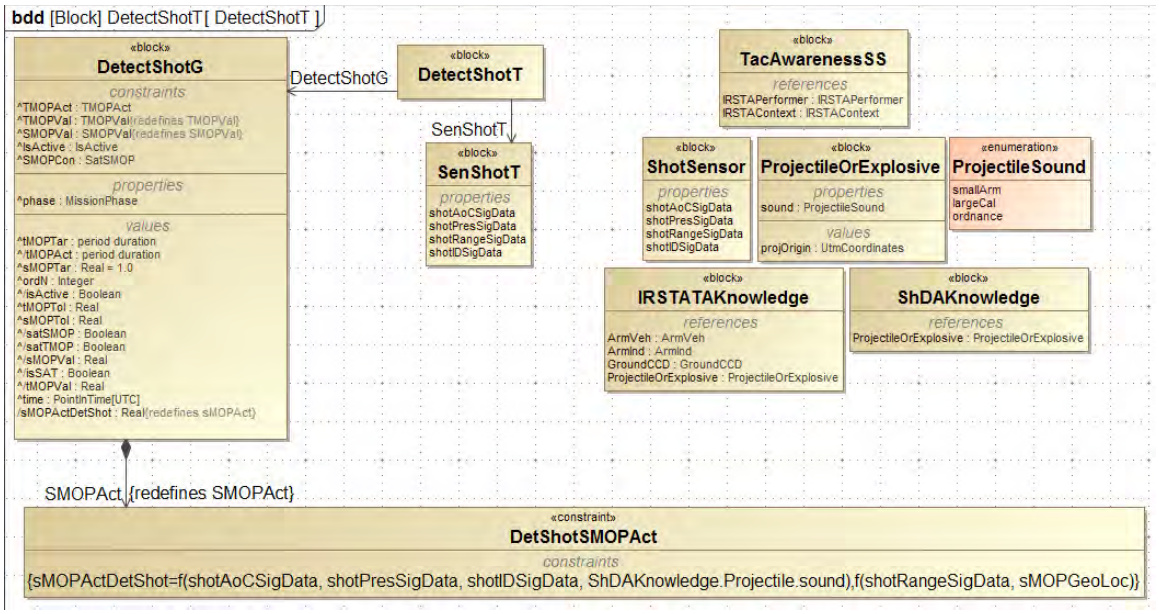

Figure 129. IRSTA Detect Shot Desired Trajectory

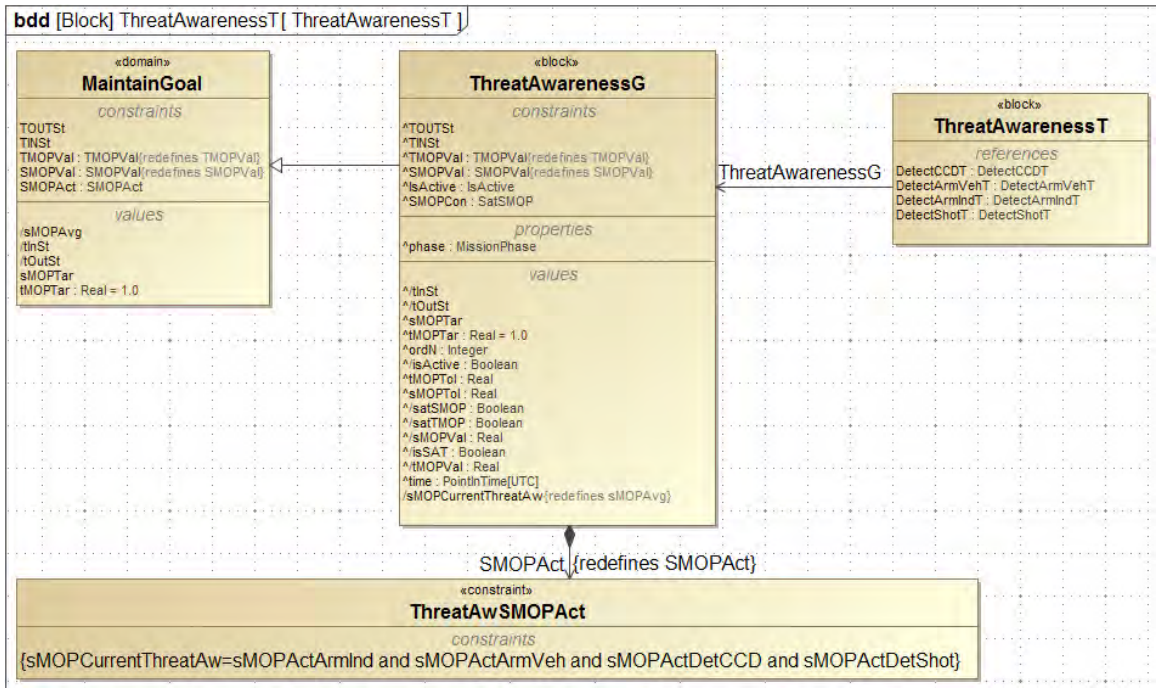

Figure 130. IRSTA Threat Awareness Desired Trajectory

2. IRSTA Logical Object Hierarchy

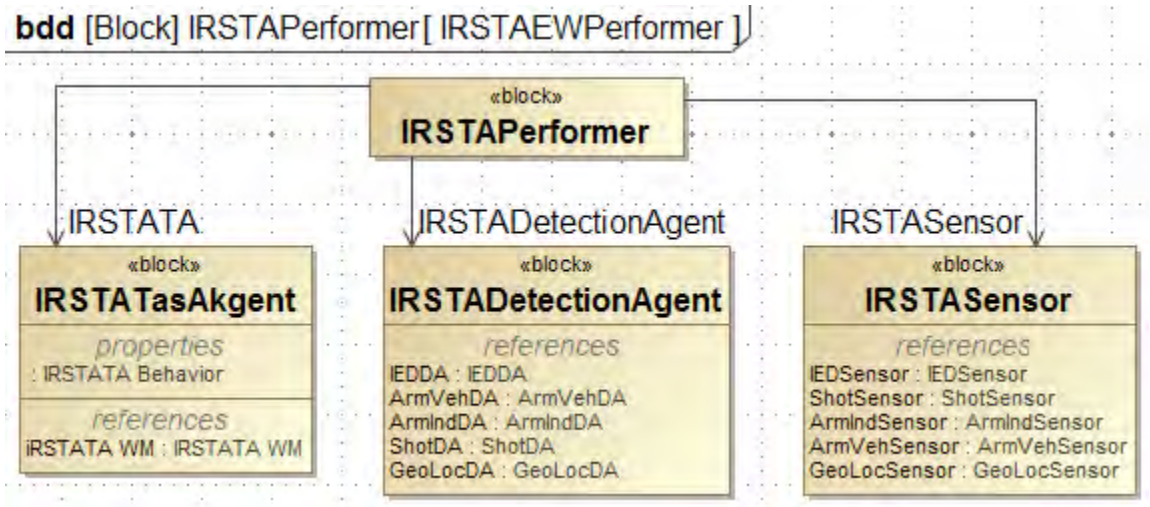

Figure 131. IRSTA Performer Logical Objects

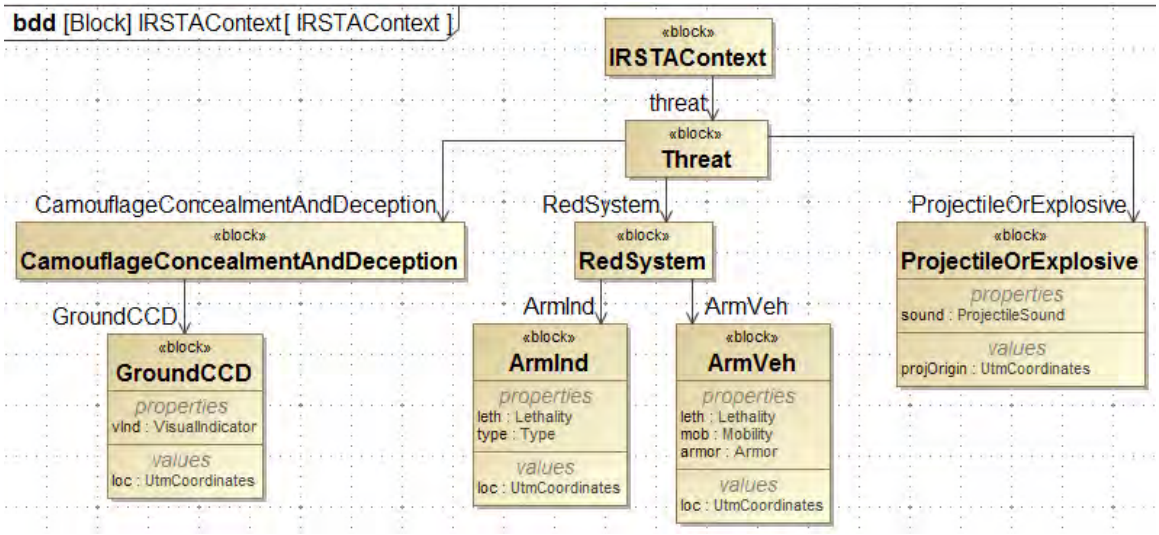

Figure 132. IRSTA Context Logical Objects

3. IRSTA Interactions

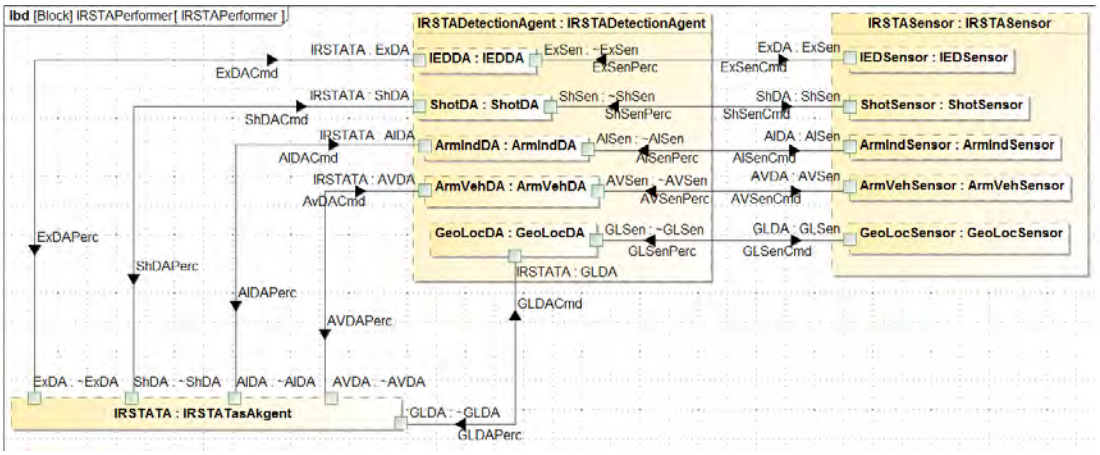

Figure 133. IRSTA Agent and Sensor Interactions

4. IRSTA Agent Internal Composition

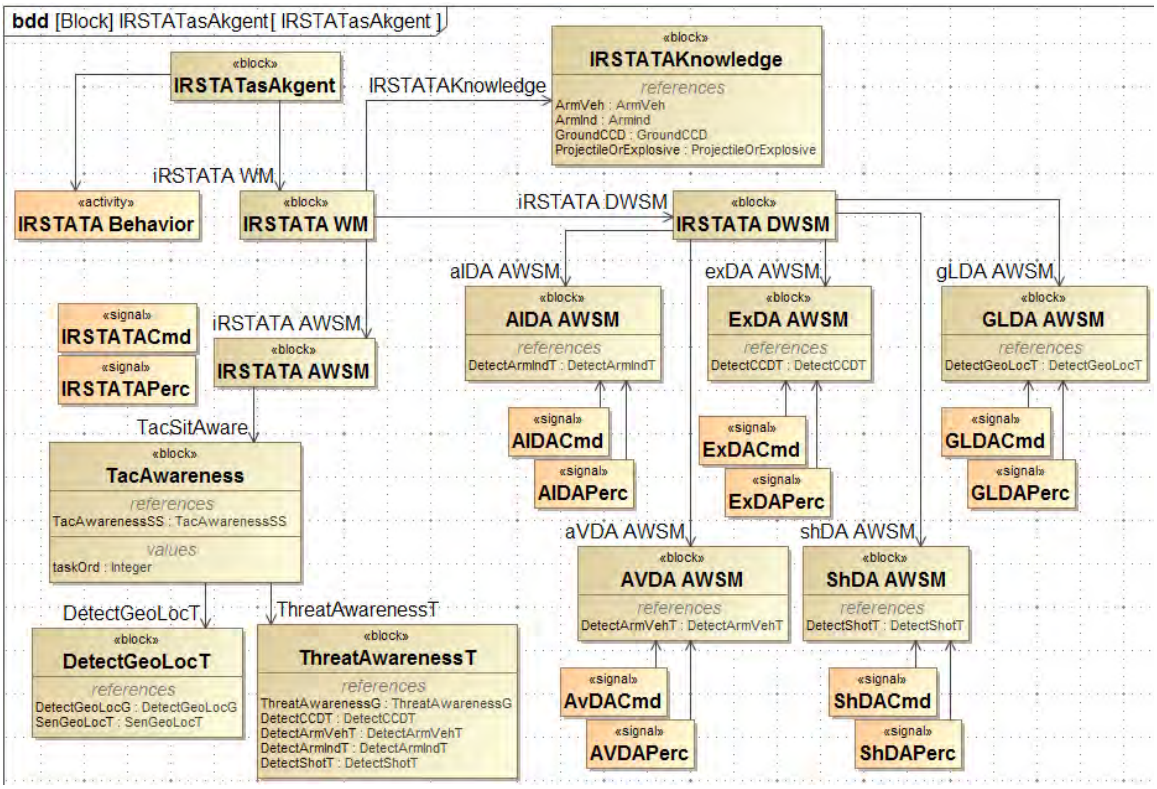

Figure 134. IRSTA Task Agent Internal Composition

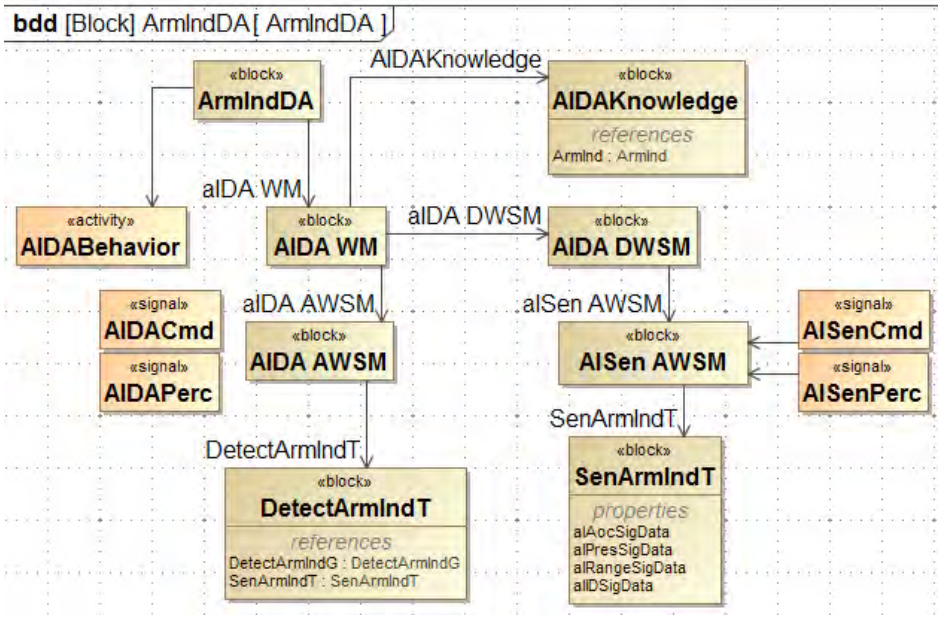

Figure 135. IRSTA Armed Individual DA Internal Composition

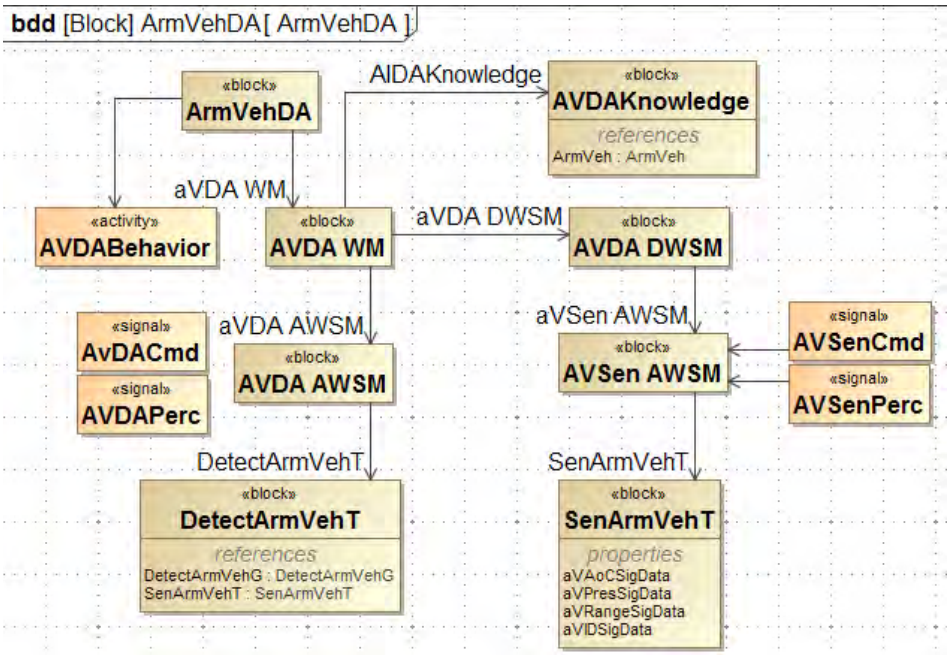

Figure 136. IRSTA Armed Vehicle DA Internal Composition

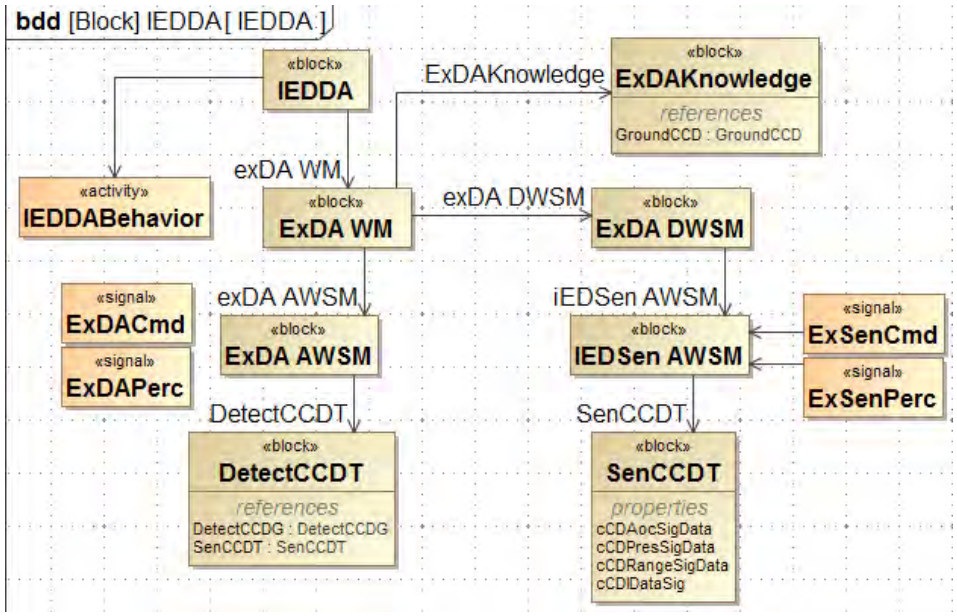

Figure 137. IRSTA IED DA Internal Composition

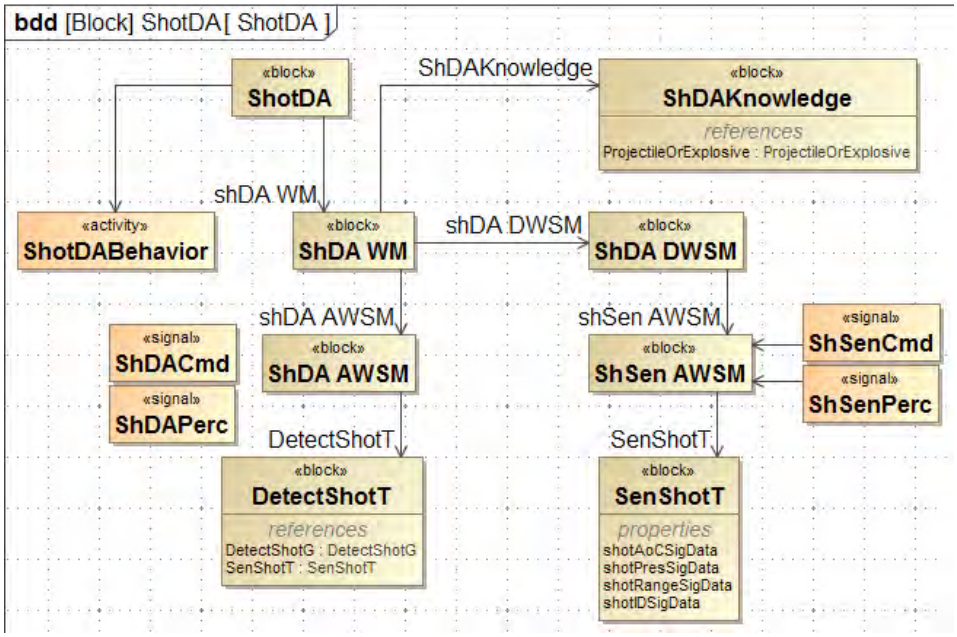

Figure 138. IRSTA Shot DA Internal Composition

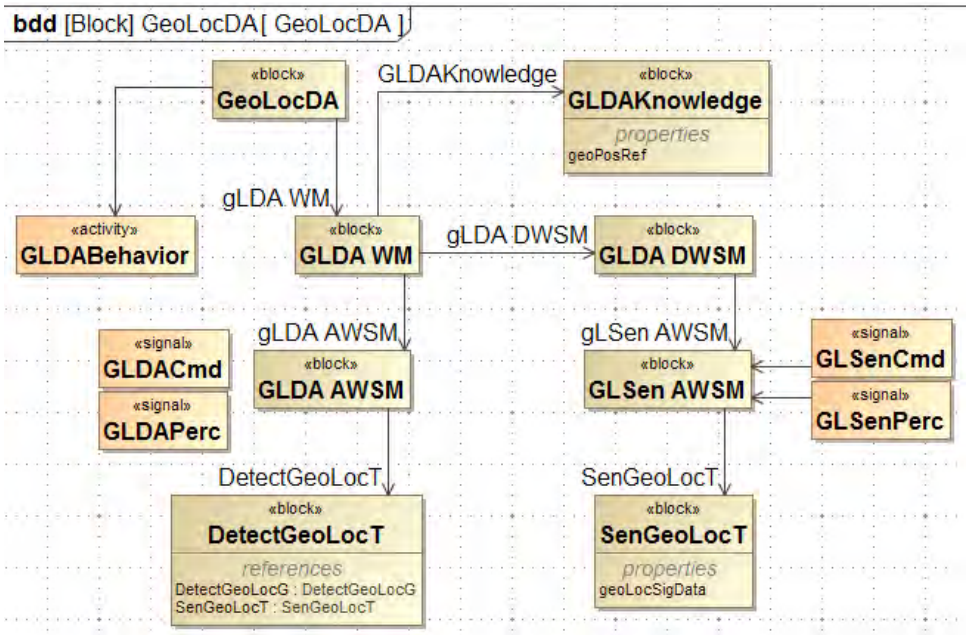

Figure 139. IRSTA Geolocation DA Internal Composition

5. IRSTA Agent Behavior

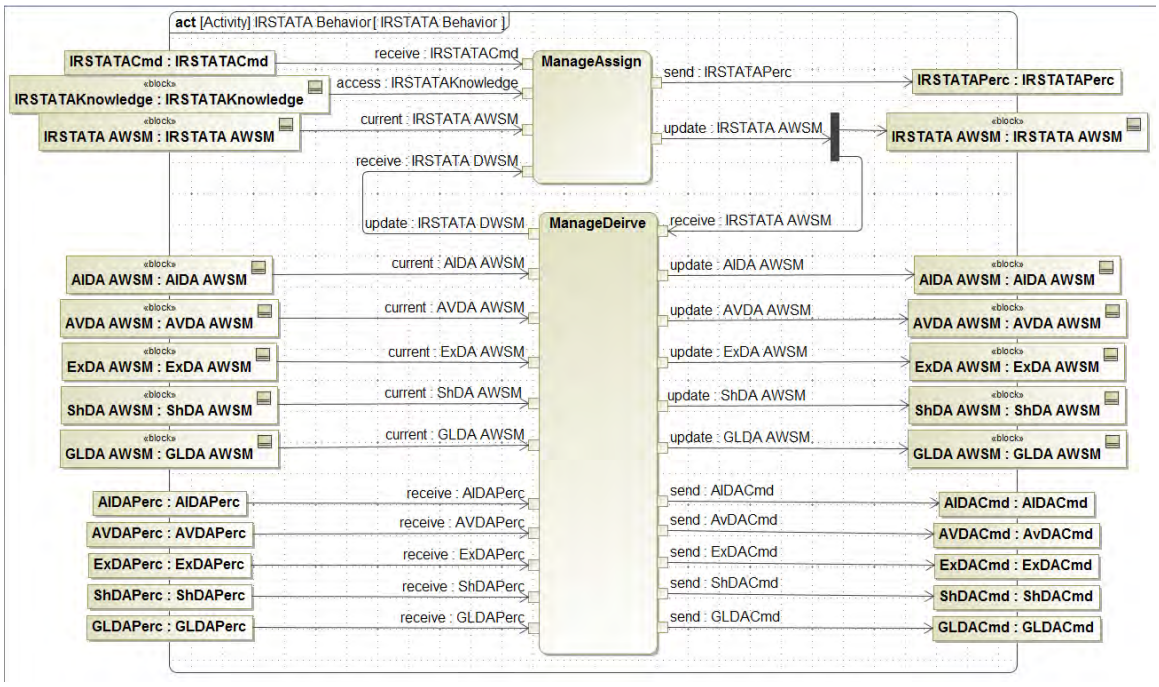

Figure 140. IRSTA Task Agent Behavior

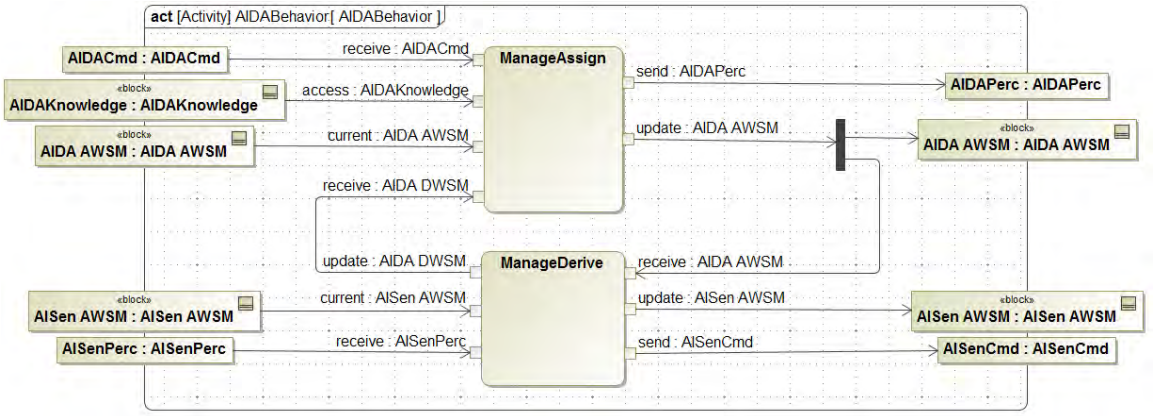

Figure 141. IRSTA Armed Individual DA Behavior

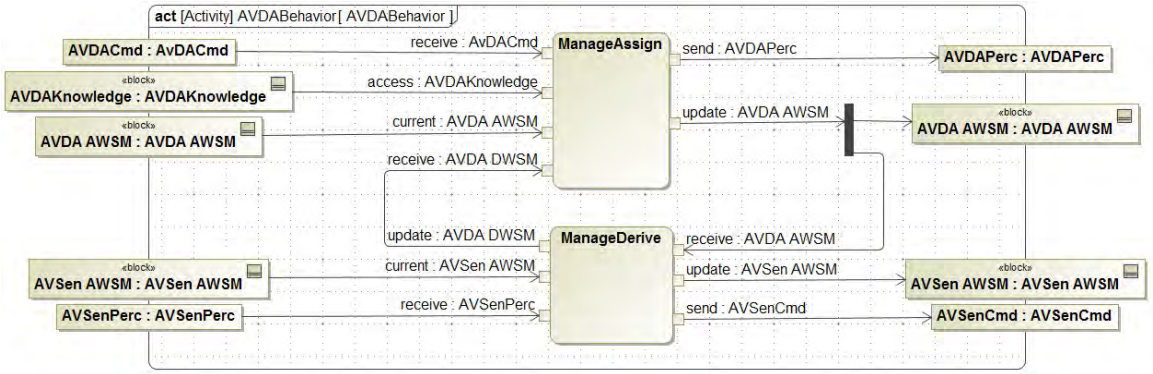

Figure 142. IRSTA Armed Vehicle DA Behavior

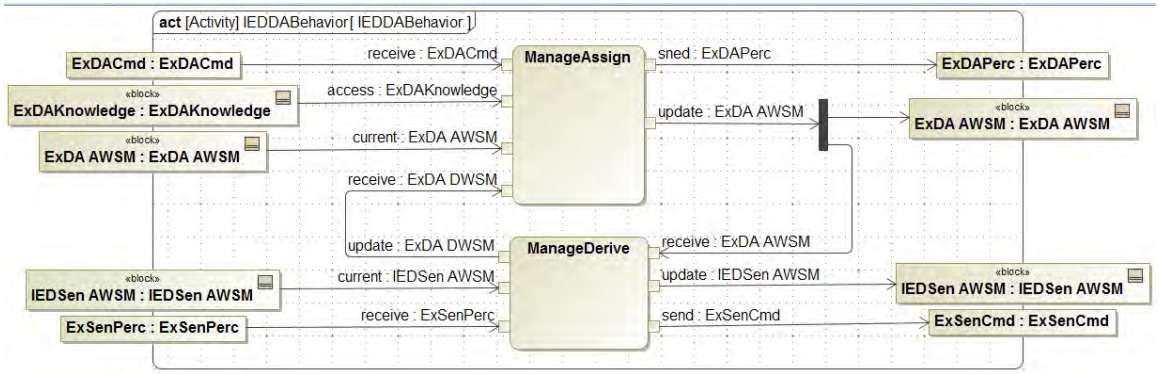

Figure 143. IRSTA IED DA Behavior

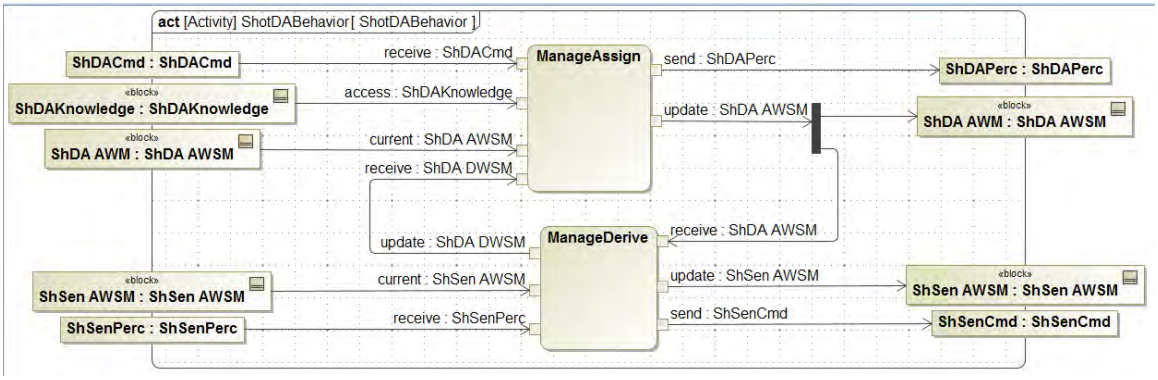

Figure 144. IRSTA Shot DA Behavior

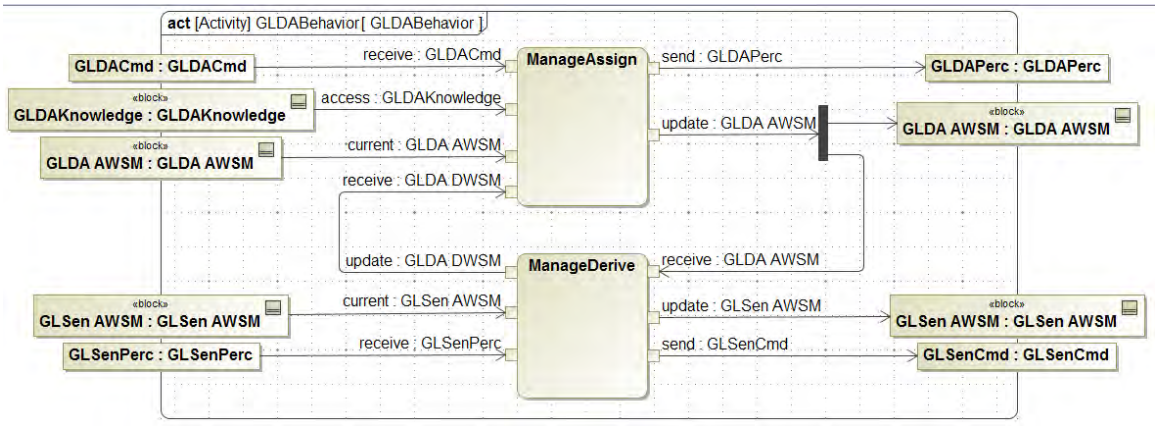

Figure 145. IRSTA Geolocation DA Behavior

E. MOBILITY

Mobility model and associated diagrams were generated by a different individual than the author as indicated in the Acknowledgment. They were generated to the concepts defined here, but the concepts were evolving and there was no SysML style guide that could be used. As such, there are some discrepancies of information captured and a different look and feel overall. Chief among these are:

(1) The variable “sAct” is used in lieu of “sMOPAct” to house constraint values. It also can contain measure of time which are separate goal measures elsewhere.

(2) Goals and trajectories are combined into a single block.

(3) A greater use of inheritance from patterns that drive a significantly different look to constraints and trajectories. The use of inheritance also requires more use of SysML "redefine" to alter or more definitely type properties that are inherited.

However, all in all, they do track to the concepts pretty closely and conform to the five modeling diagram types.

1. Mobility Assigned Mission

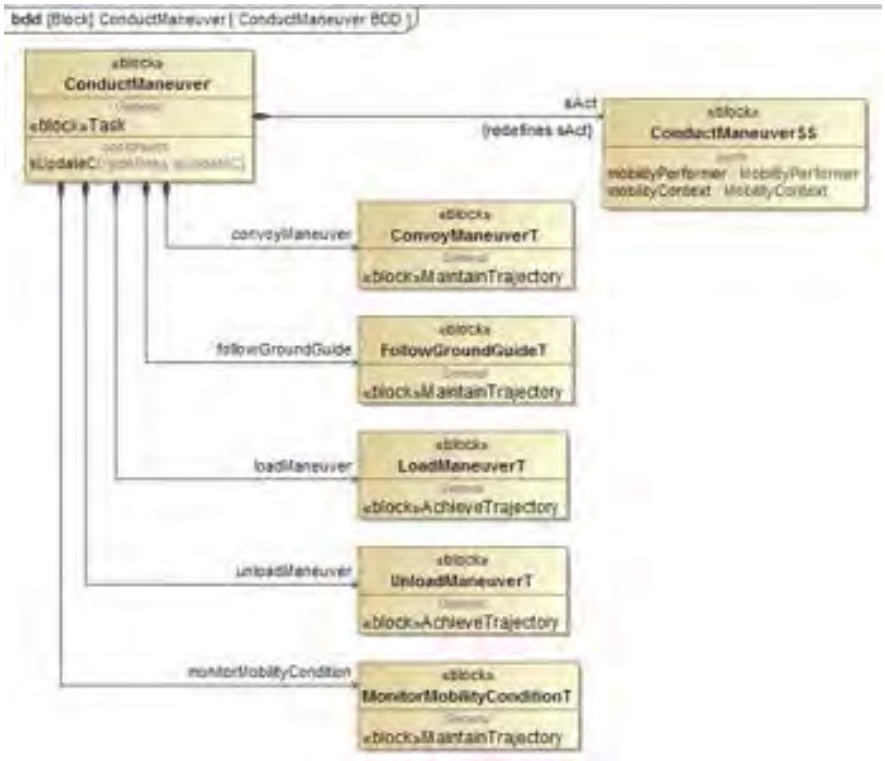

Figure 146. Mobility Conduct Maneuver Task

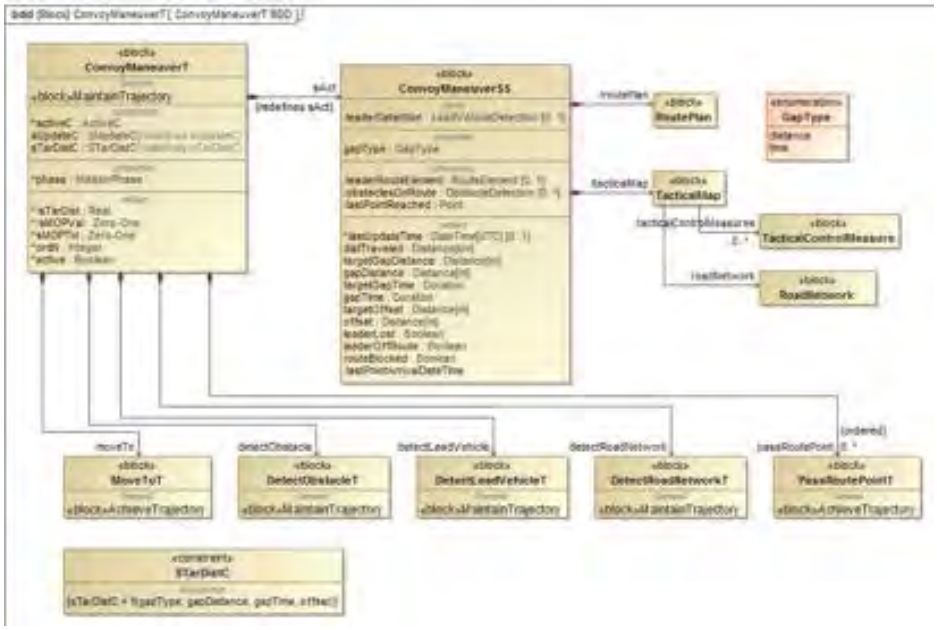

Figure 147. Mobility Convoy Maneuver Desired Trajectory

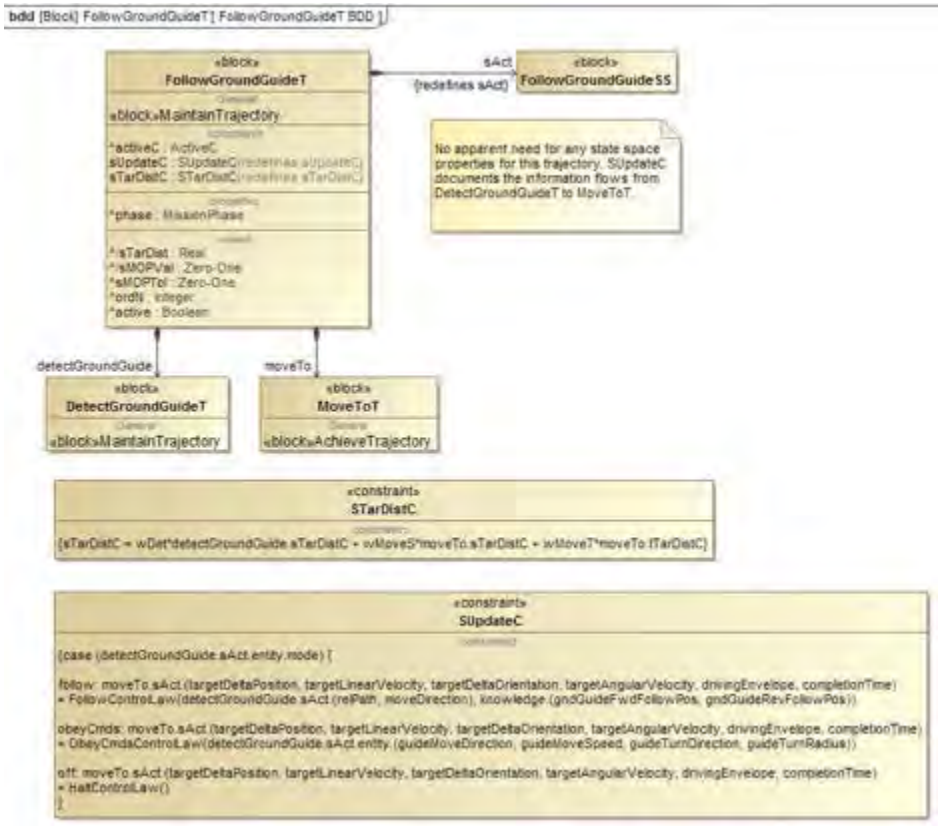

Figure 148. Mobility Follow Ground Guide Desired Trajectory

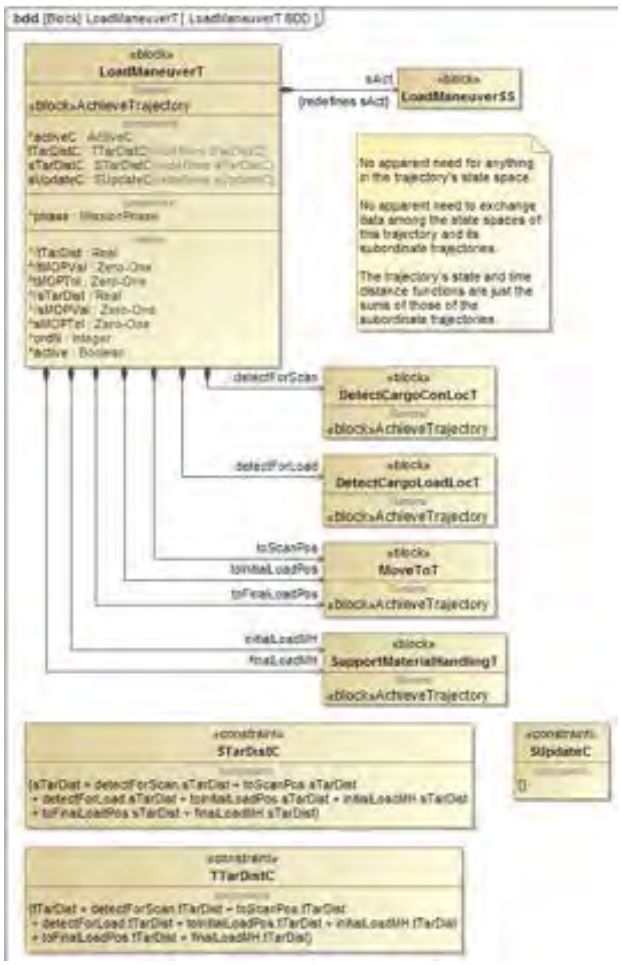

Figure 149. Mobility Load Maneuver Desired Trajectory

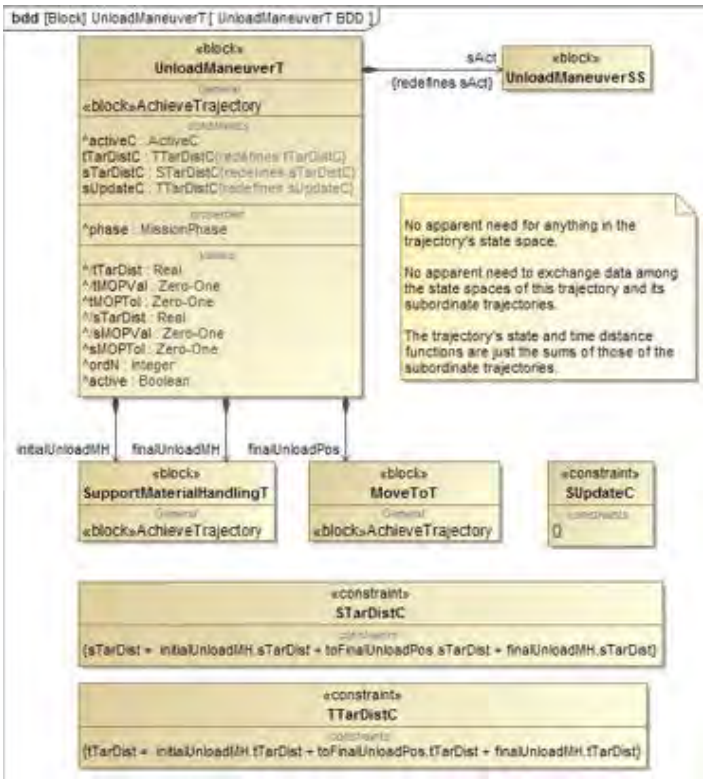

Figure 150. Mobility Unload Maneuver Desired Trajectory

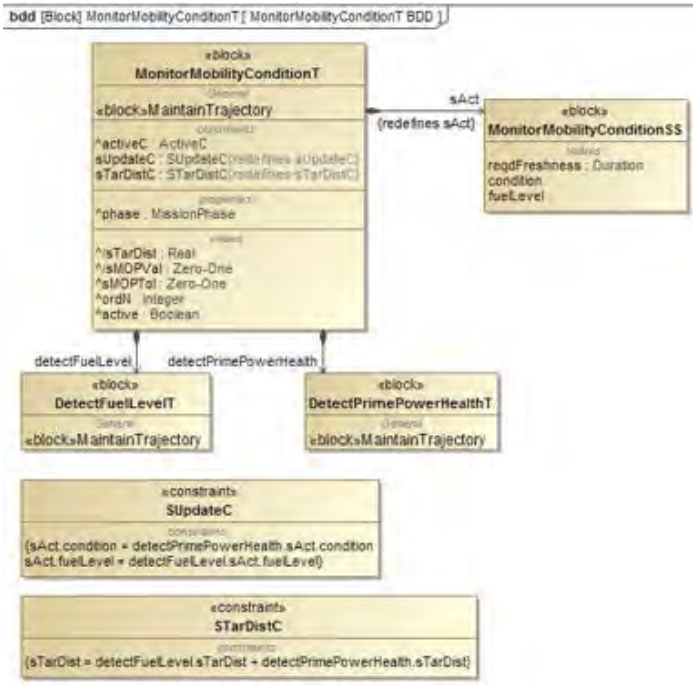

Figure 151. Mobility Monitor Mobility Condition Desired Trajectory

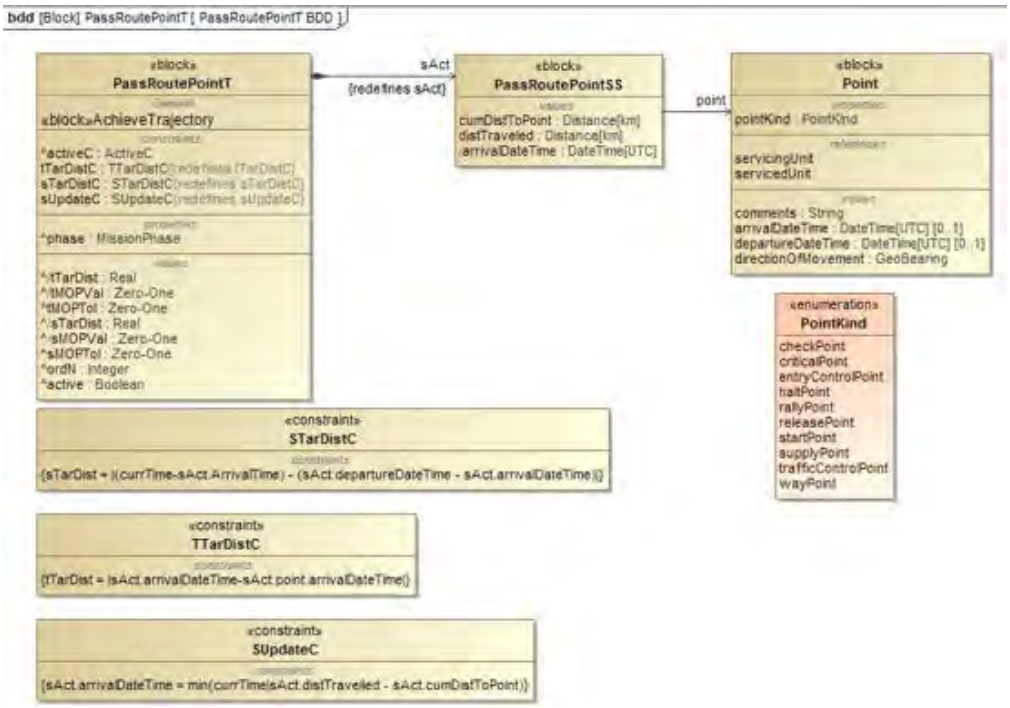

Figure 152. Mobility Pass Route Point Desired Trajectory

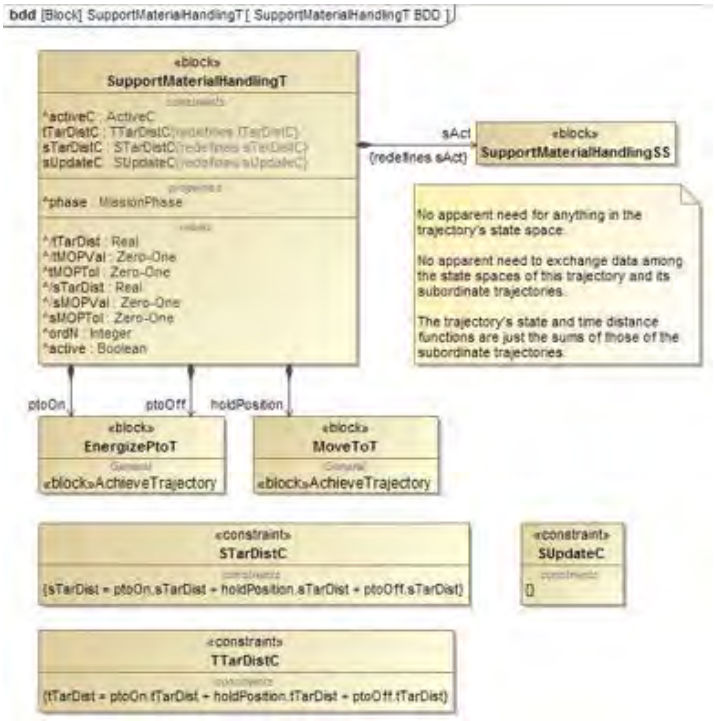

Figure 153. Mobility Support Material Handling Desired Trajectory

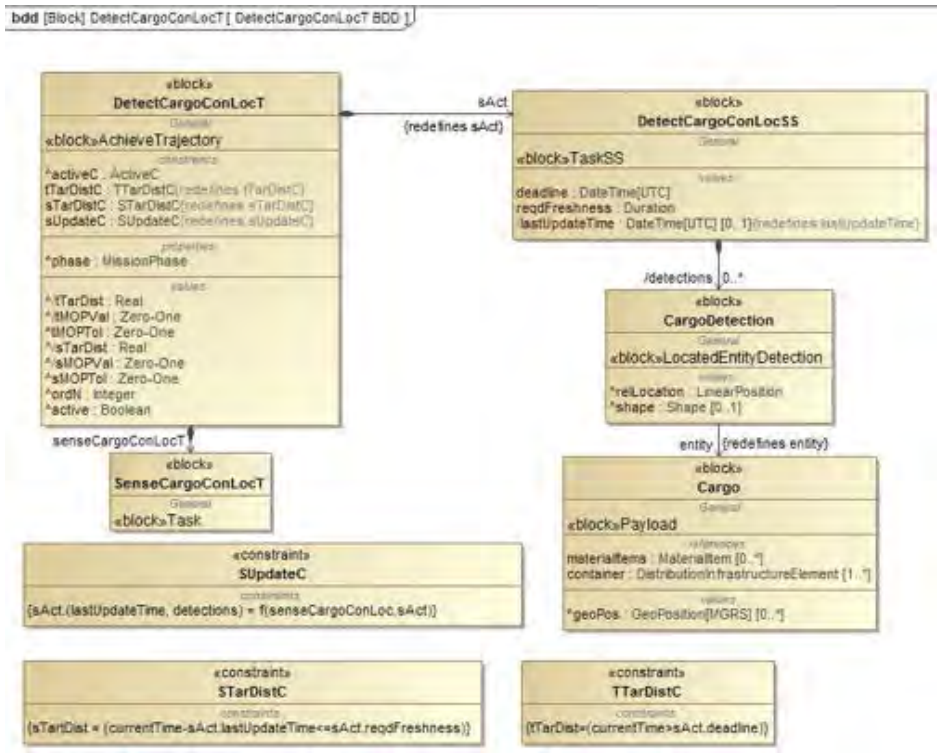

Figure 154. Mobility Detect Cargo Content Location

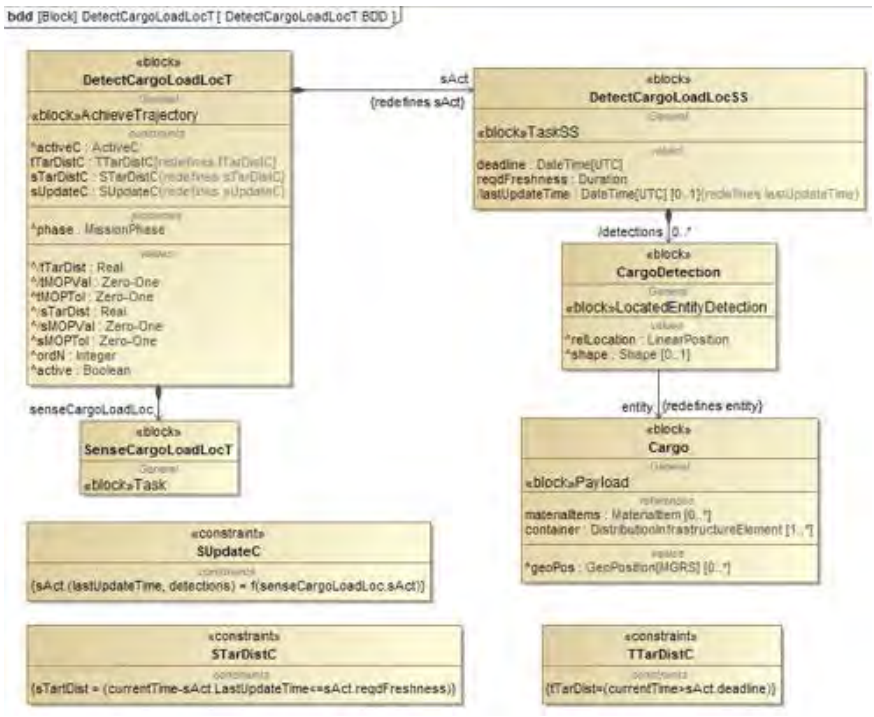

Figure 155. Mobility Detect Cargo Load Location Desired Trajectory

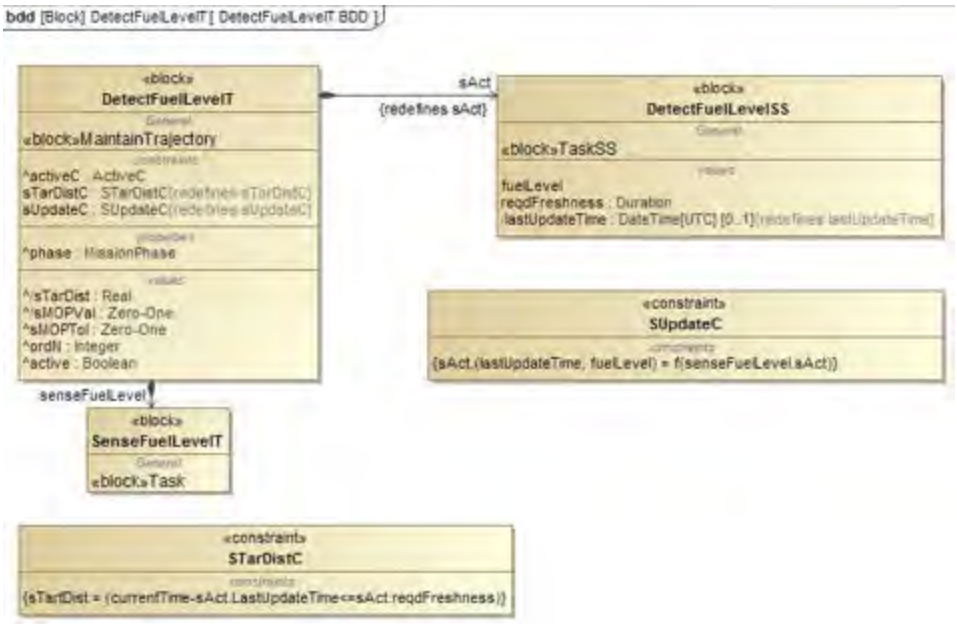

Figure 156. Mobility Detect Fuel Level Desired Trajectory

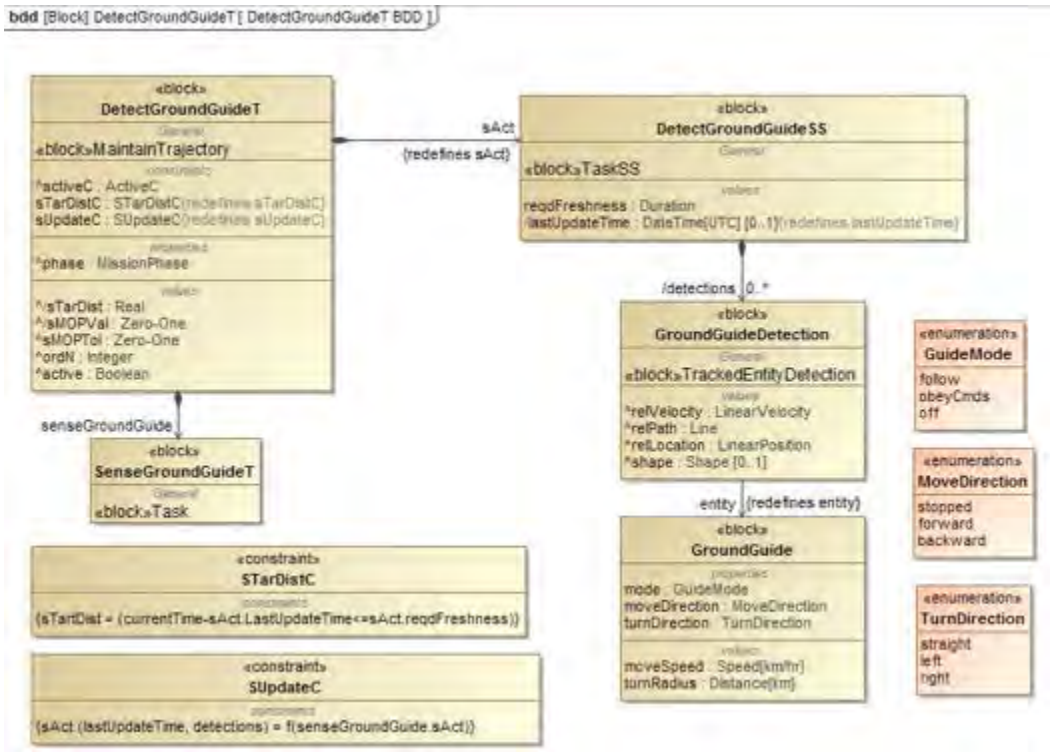

Figure 157. Mobility Detect Ground Guide Desired Trajectory

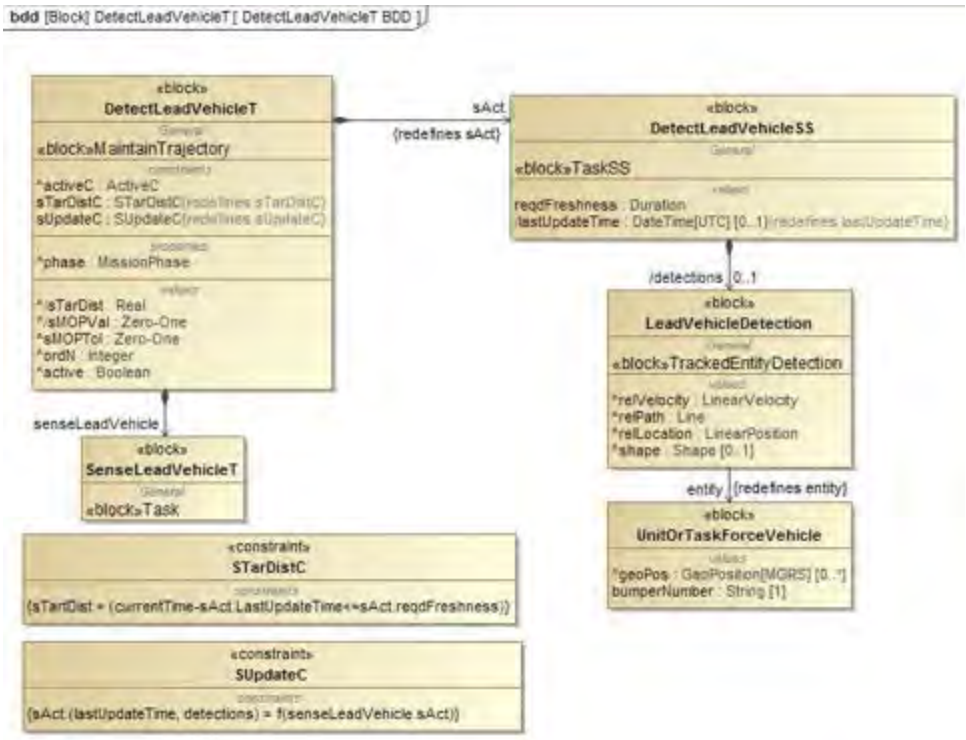

Figure 158. Mobility Detect Lead Vehicle Desired Trajectory

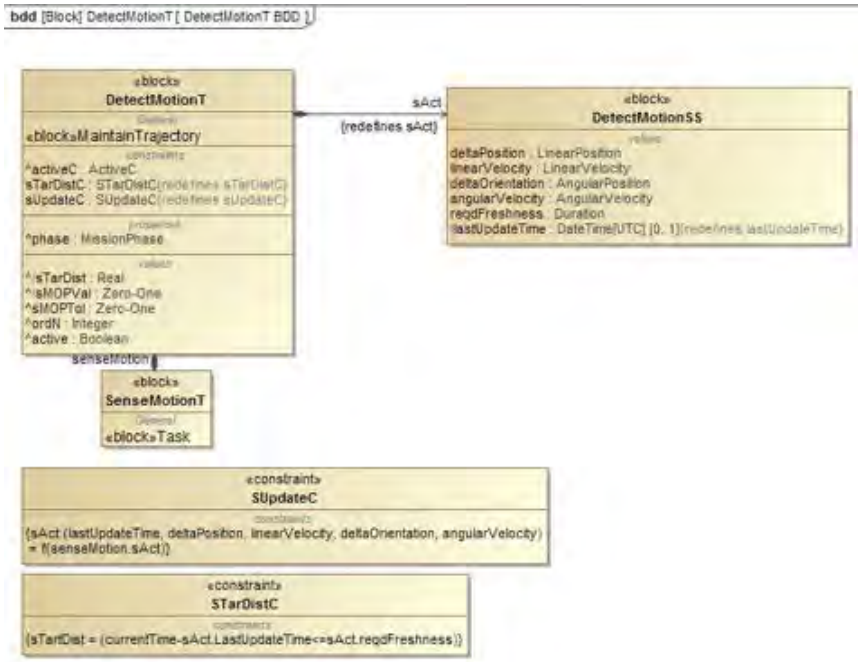

Figure 159. Mobility Detect Motion Desired Trajectory

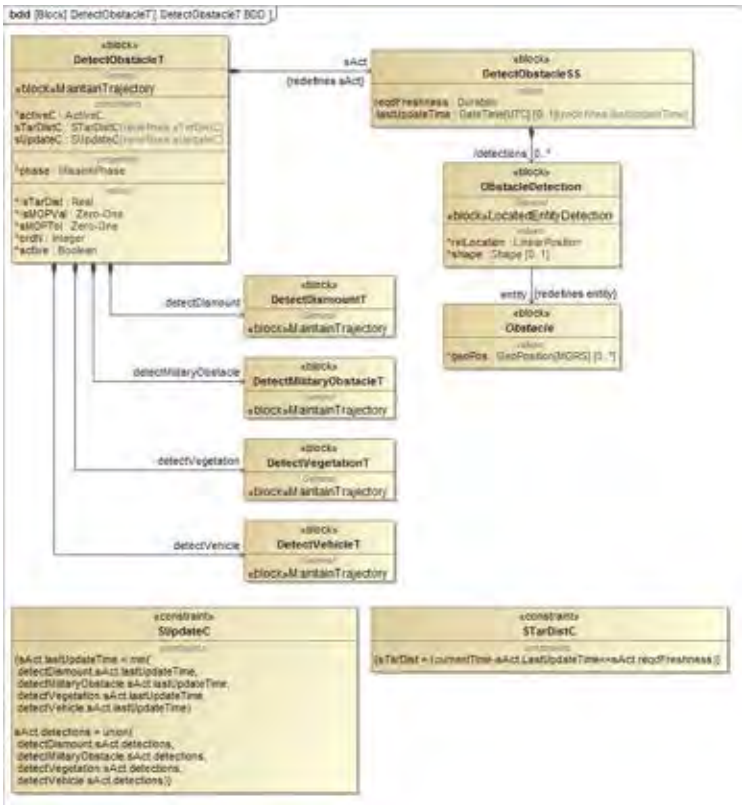

Figure 160. Mobility Detect Obstacle Desired Trajectory

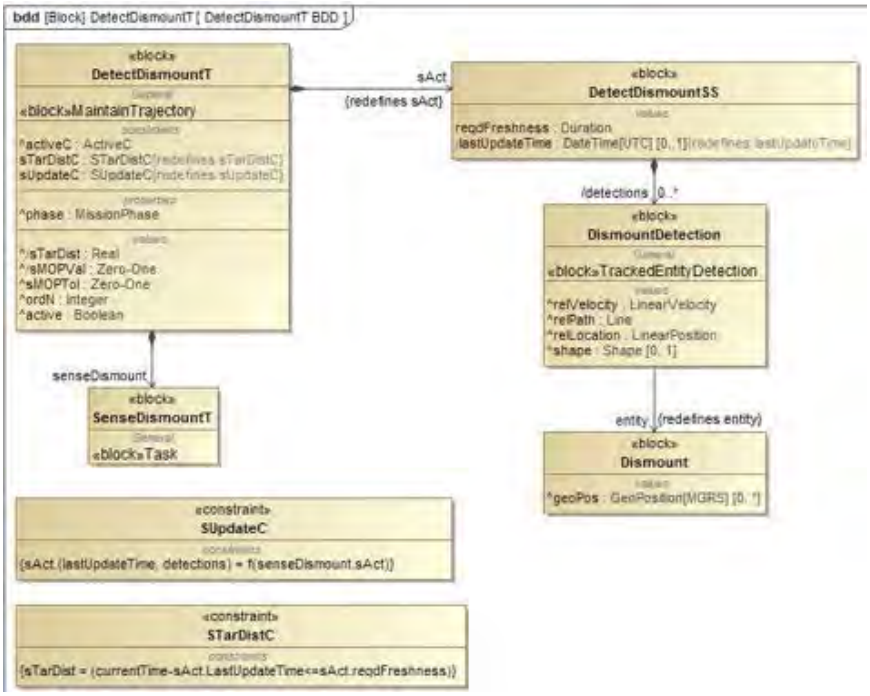

Figure 161. Mobility Detect Dismount Desired Trajectory

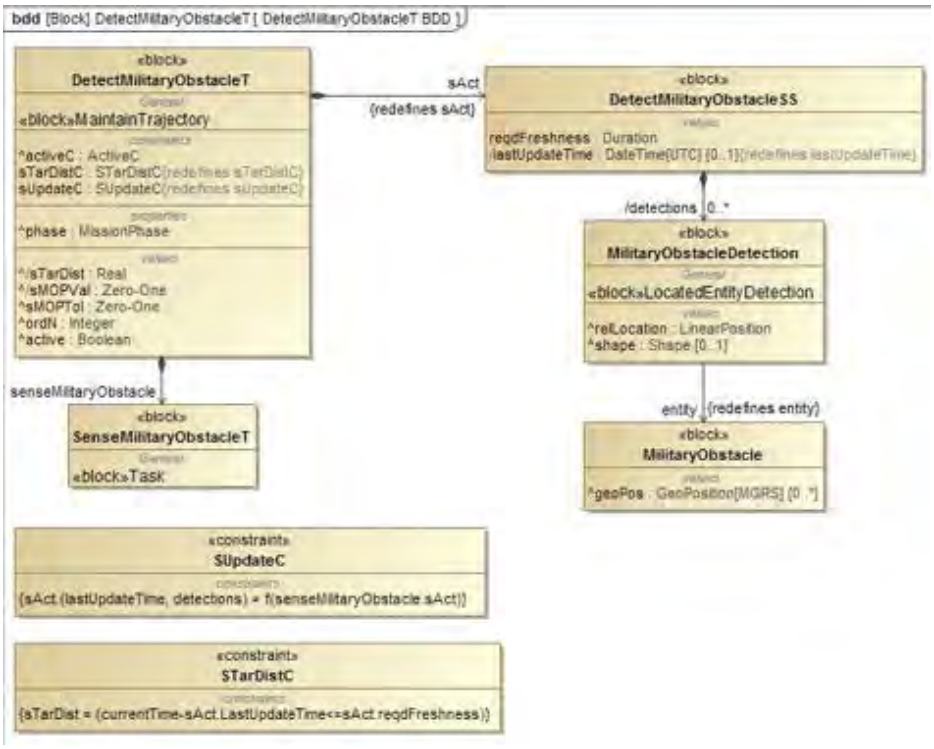

Figure 162. Mobility Detect Military Obstacle Desired Trajectory

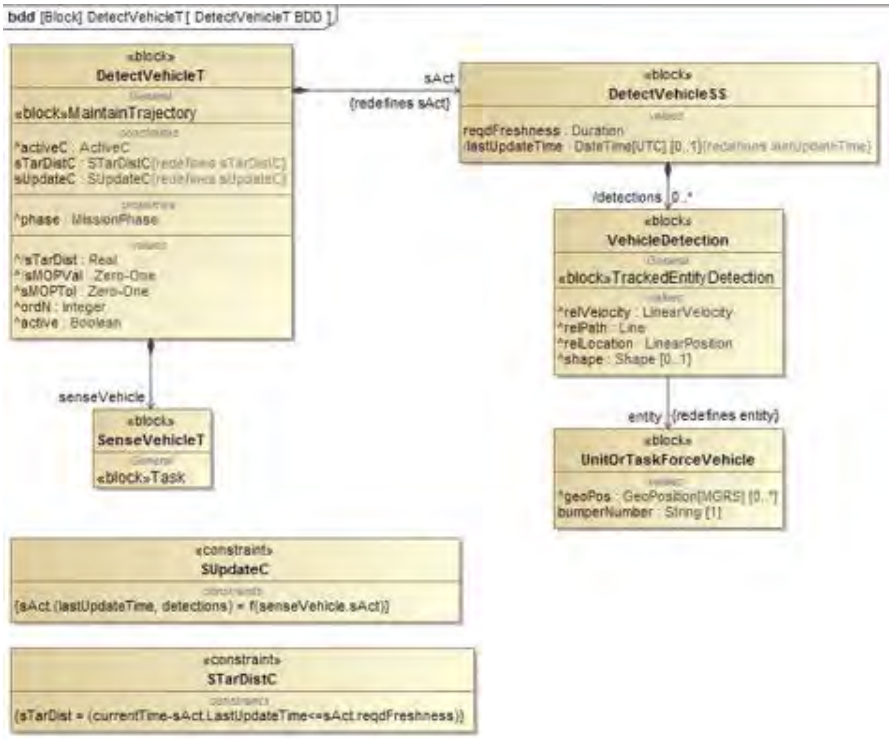

Figure 163. Mobility Detect Vehicle Desired Trajectory

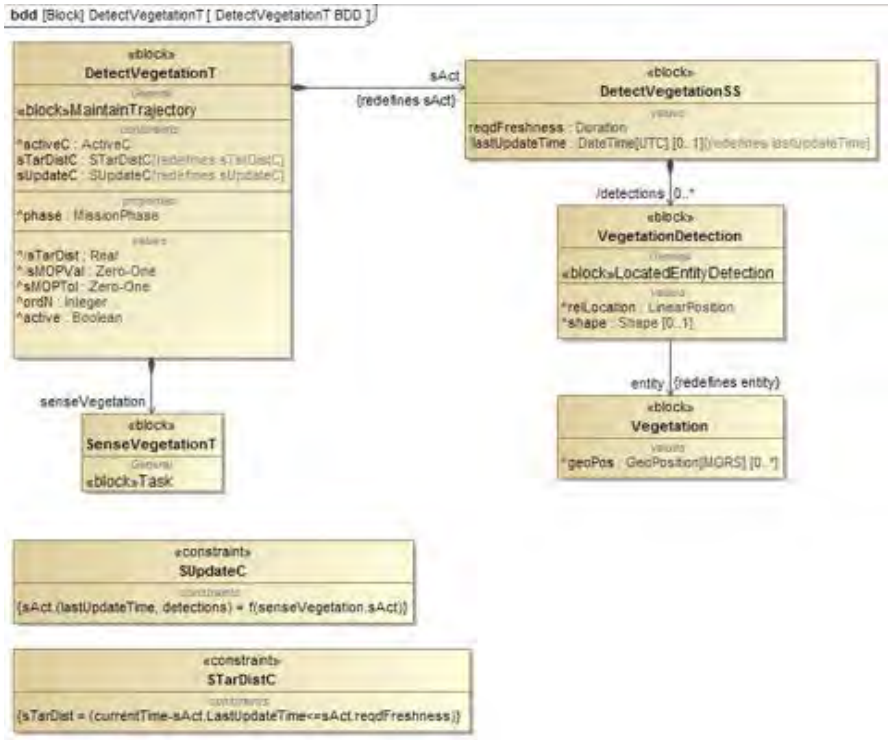

Figure 164. Mobility Detect Vegetation Desired Trajectory

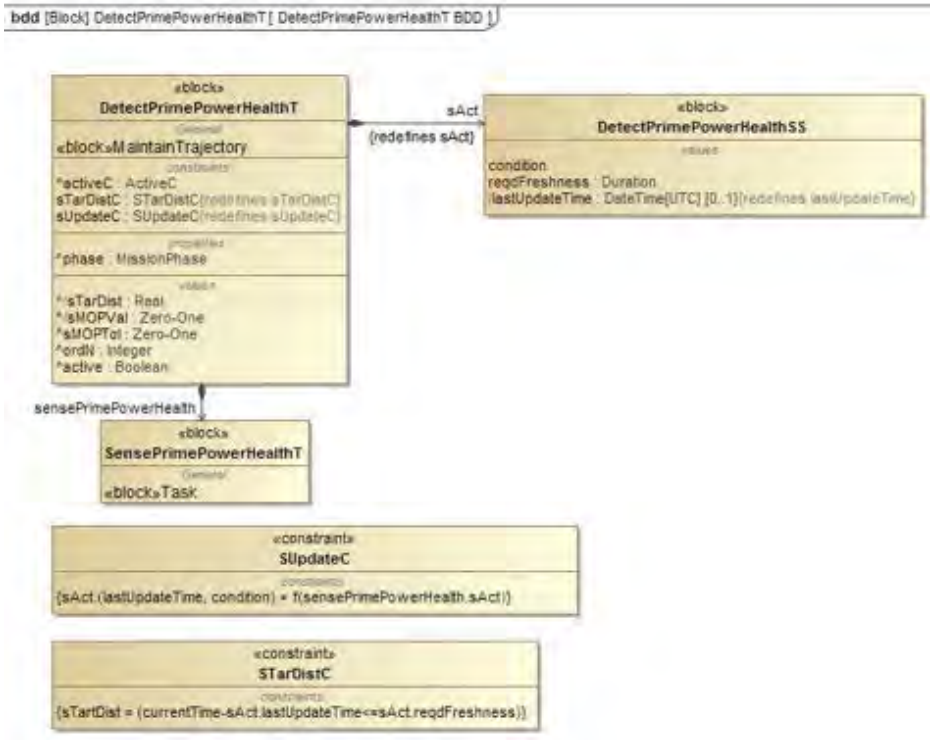

Figure 165. Mobility Detect Prime Power Health Desired Trajectory

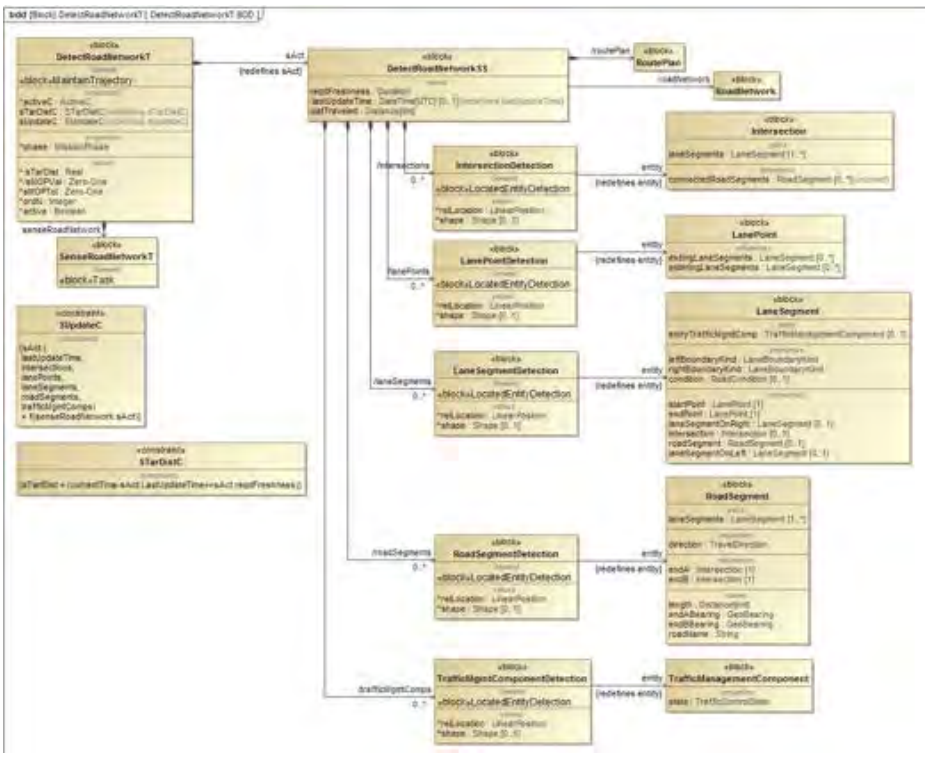

Figure 166. Mobility Detect Road Network Desired Trajectory

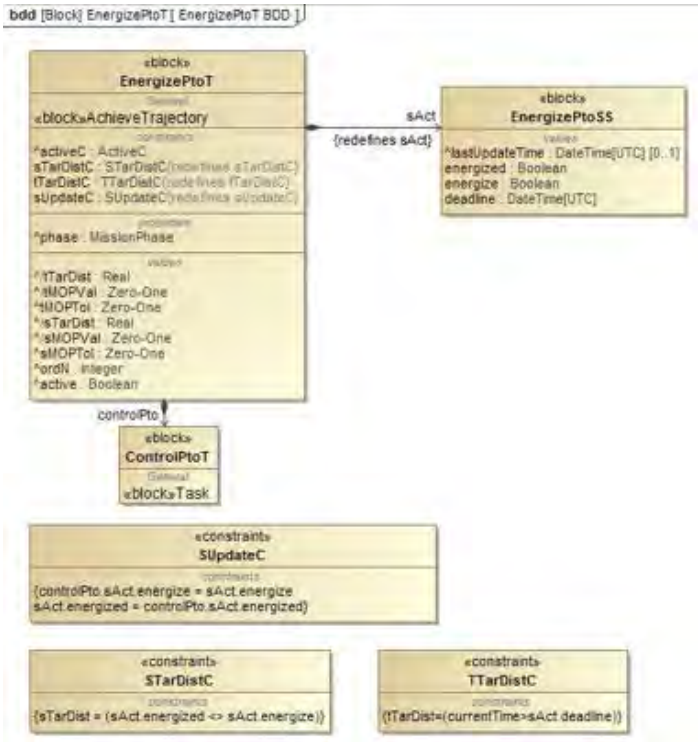

Figure 167. Mobility Energize PTO Desired Trajectory

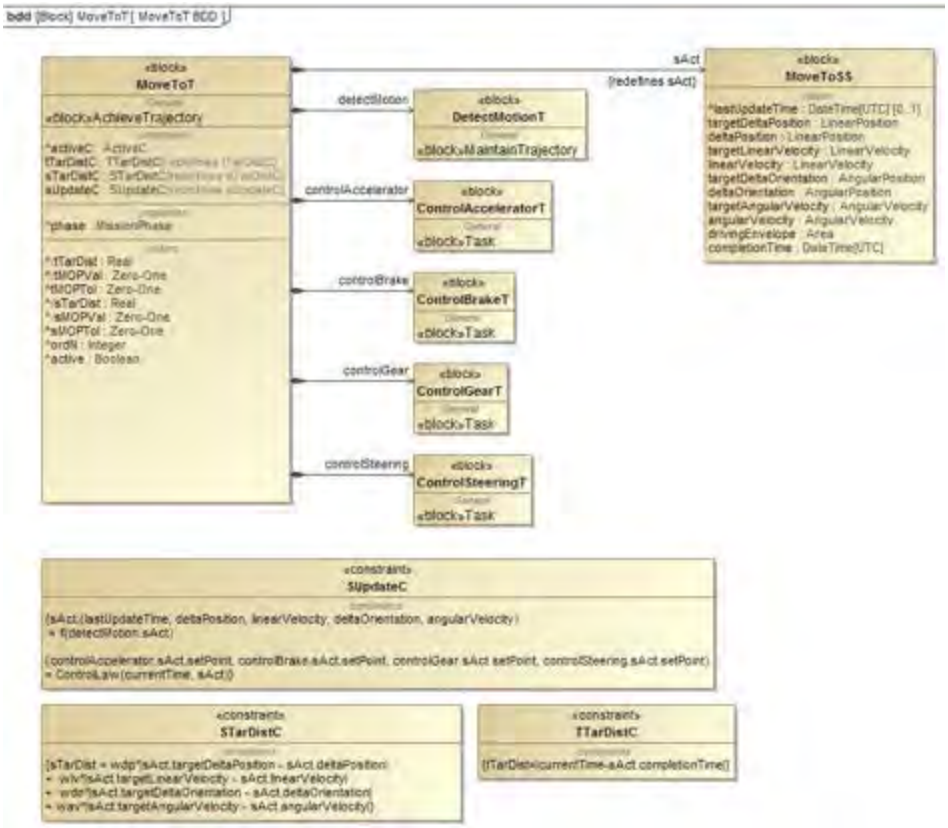

Figure 168. Mobility “Move To” Desired Trajectory

2. Mobility Logical Object Hierarchy

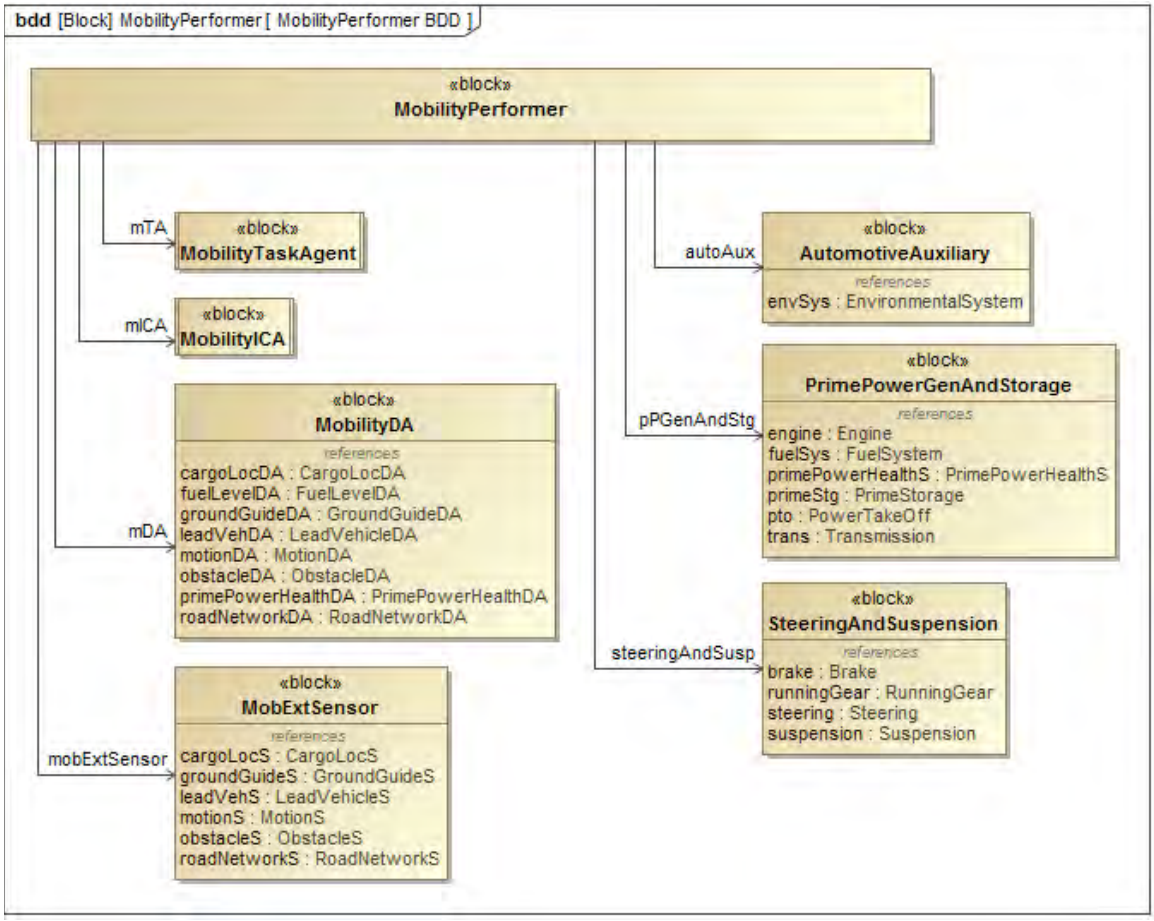

Figure 169. Mobility Performer Logic Objects

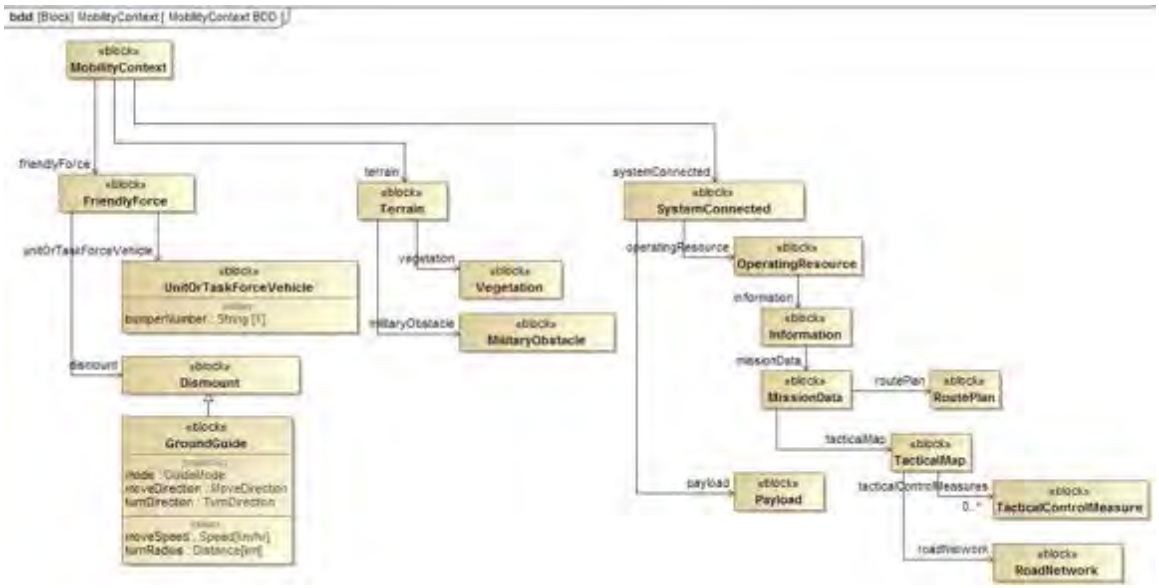

Figure 170. Mobility Context Logical Objects

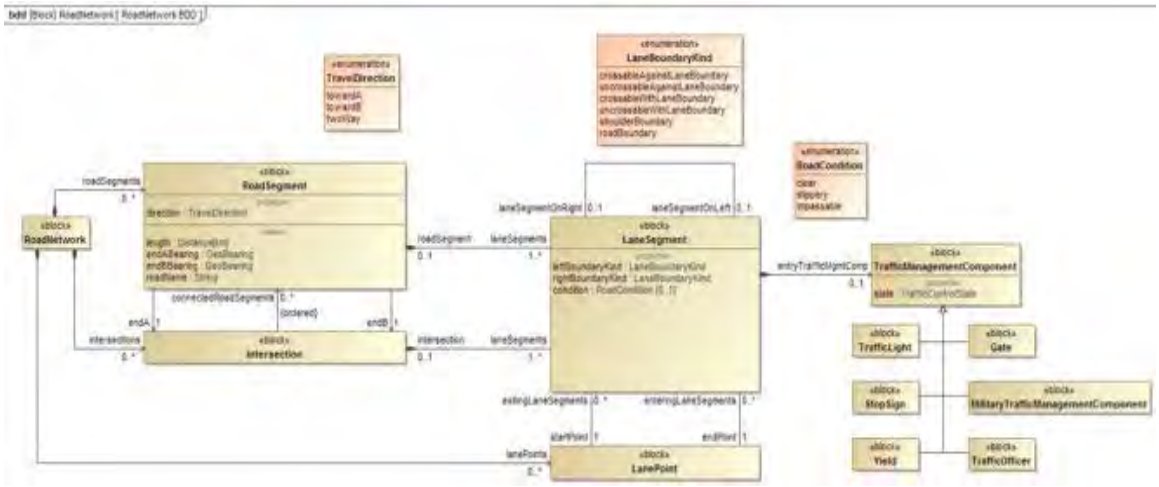

Figure 171. Mobility Context Road Network Logical Objects

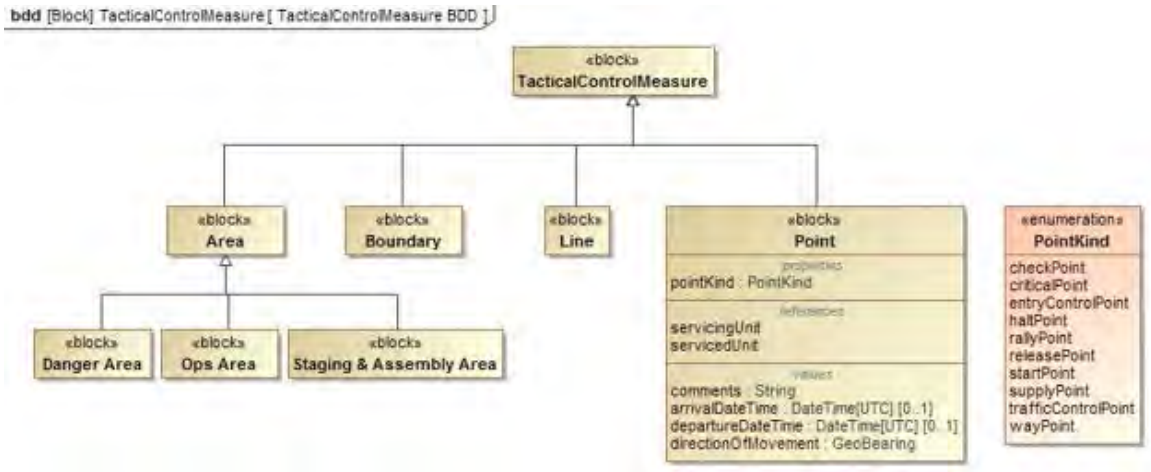

Figure 172. Mobility Context Tactical Control Measures Logical Object

3. Mobility Interactions

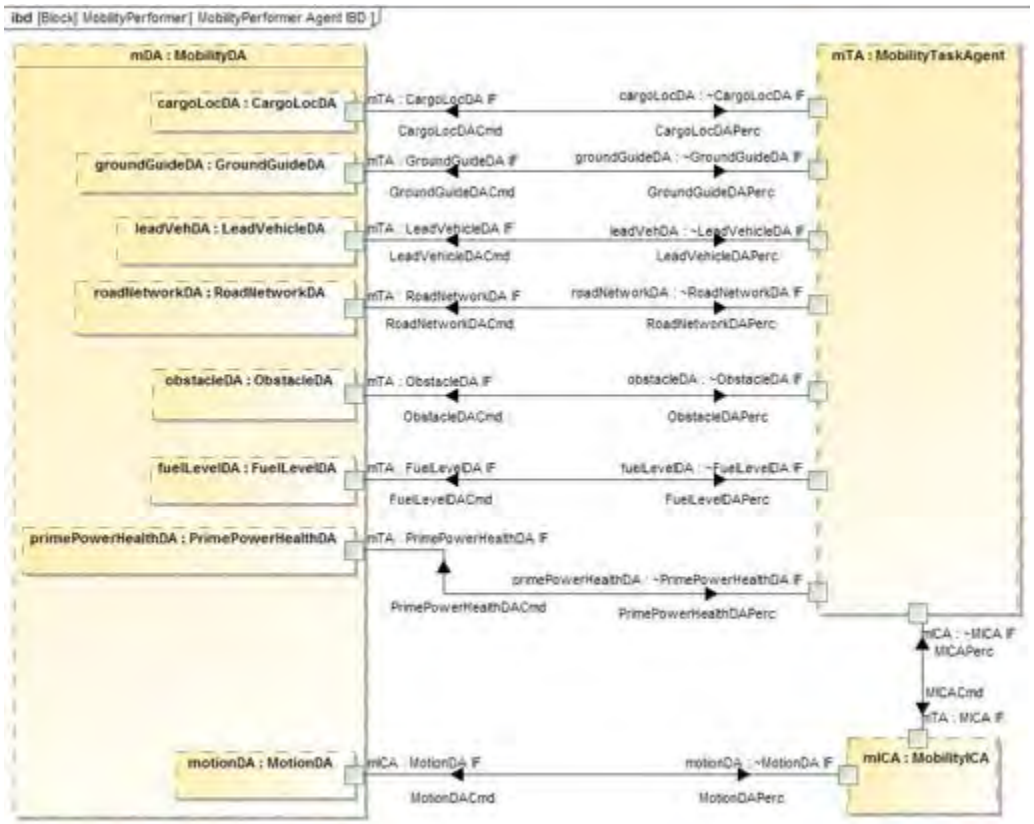

Figure 173. Mobility Agent Interactions

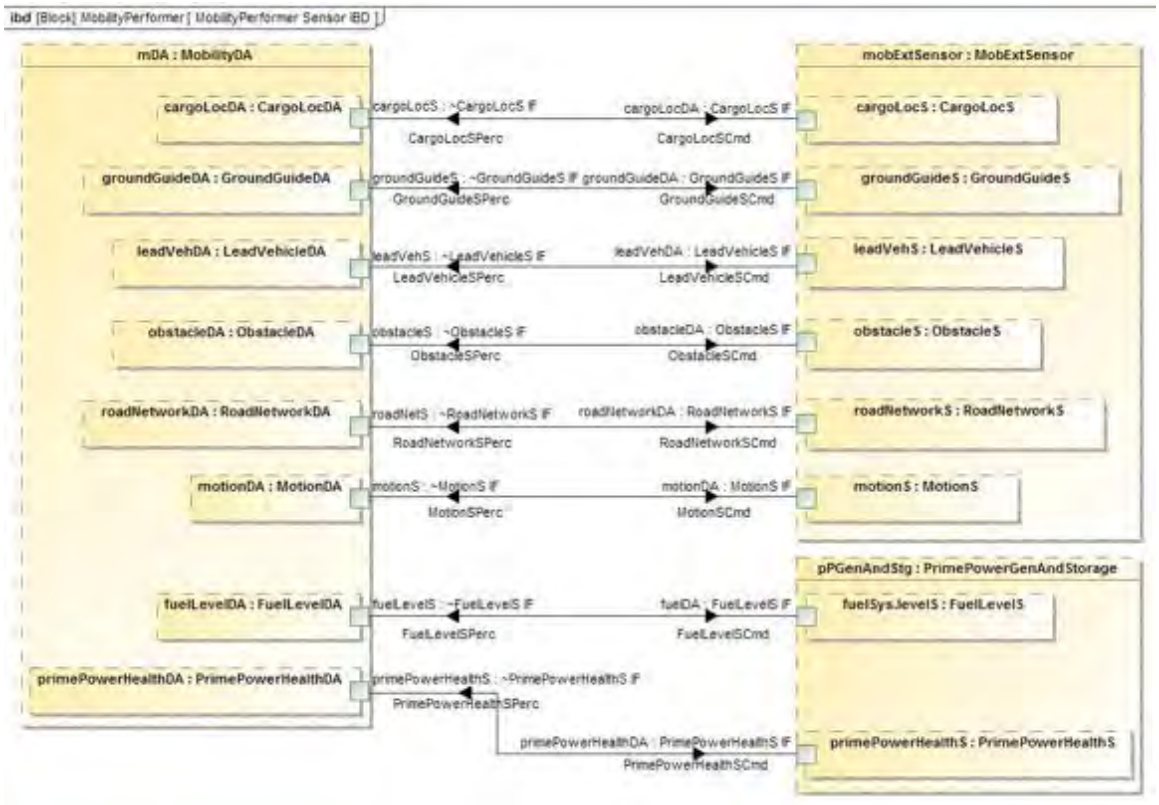

Figure 174. Mobility DA and Sensor Interactions

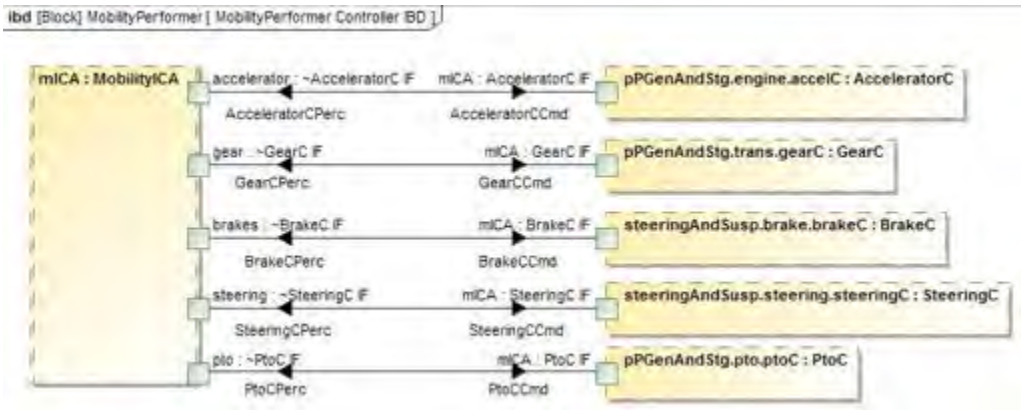

Figure 175. Mobility ICA and Controller Interactions

4. Mobility Agent Internal Composition

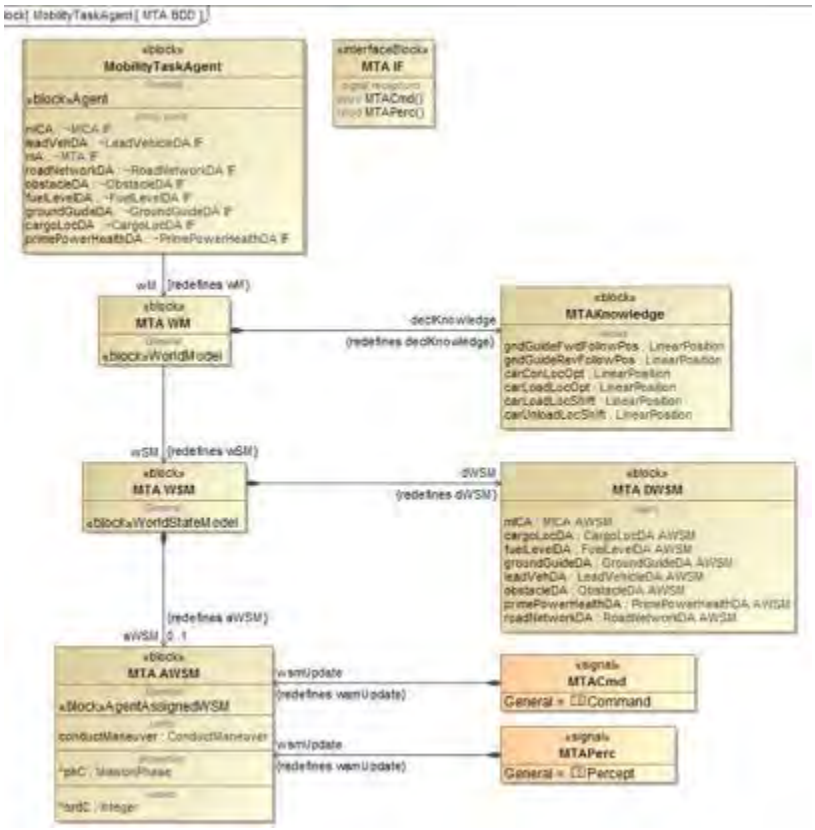

Figure 176. Mobility Task Agent Internal Composition

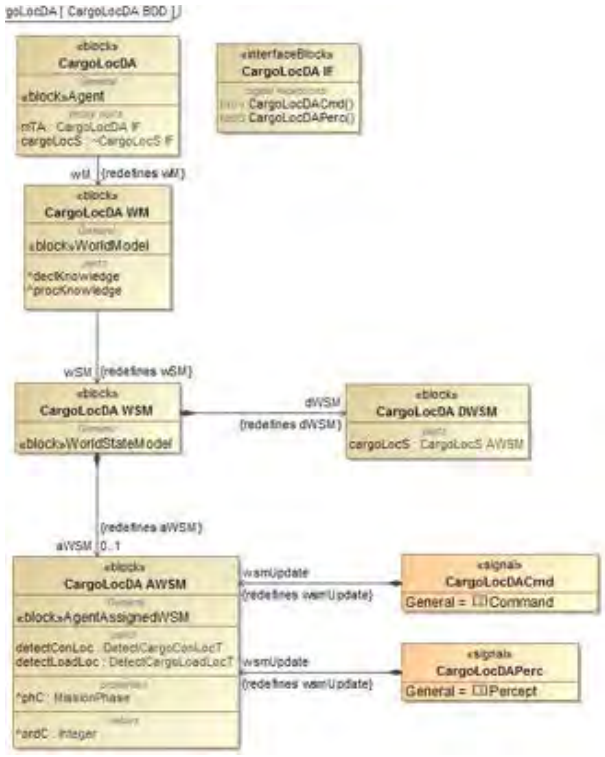

Figure 177. Mobility Cargo DA Internal Composition

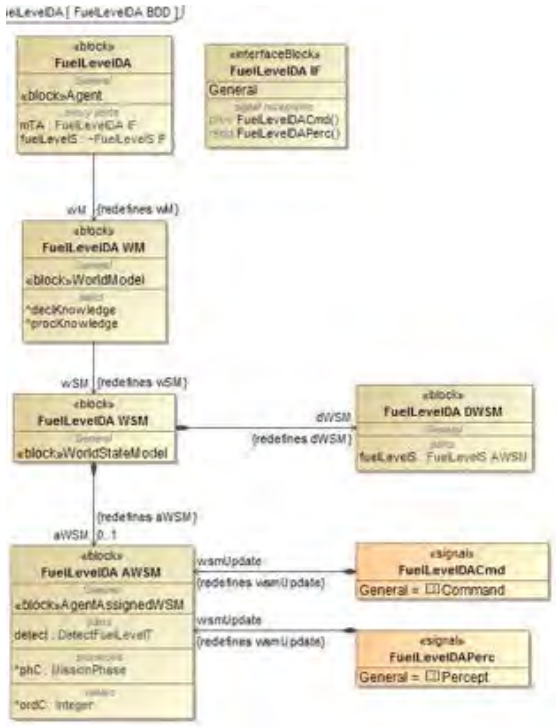

Figure 178. Mobility Fuel Level DA Internal Composition

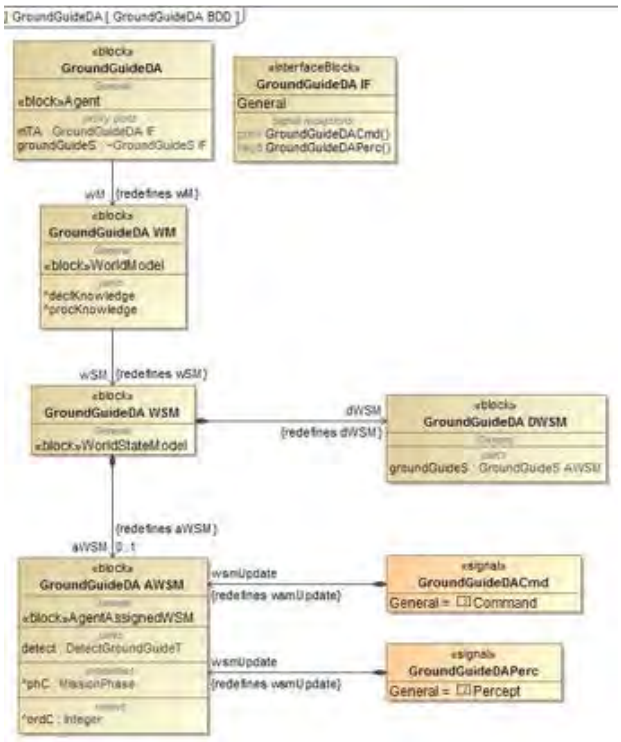

Figure 179. Mobility Ground Guide DA Internal Composition

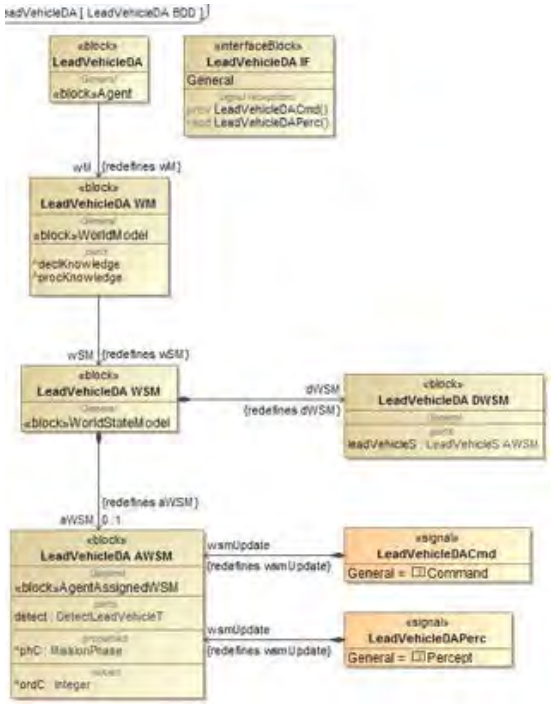

Figure 180. Mobility Lead Vehicle DA internal Composition

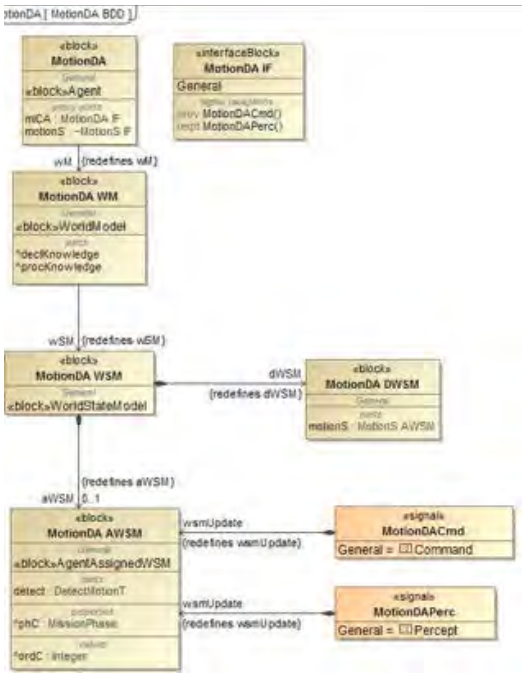

Figure 181. Mobility Motion DA Internal Composition

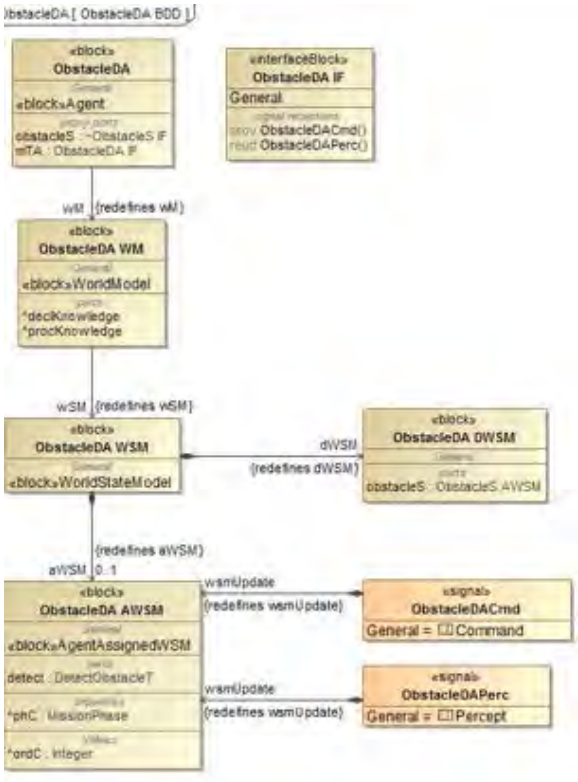

Figure 182. Mobility Obstacle DA Internal Composition

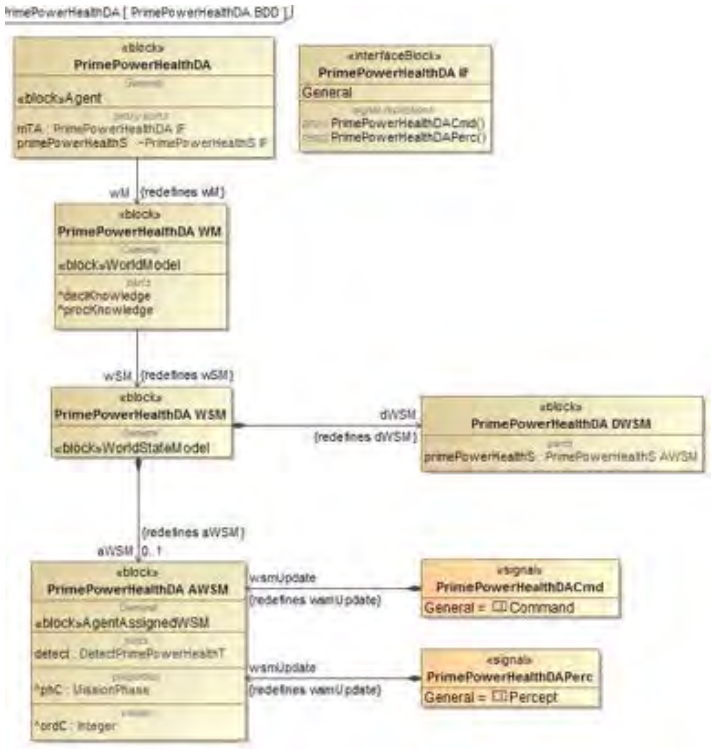

Figure 183. Mobility Primer Power Health DA Internal Composition

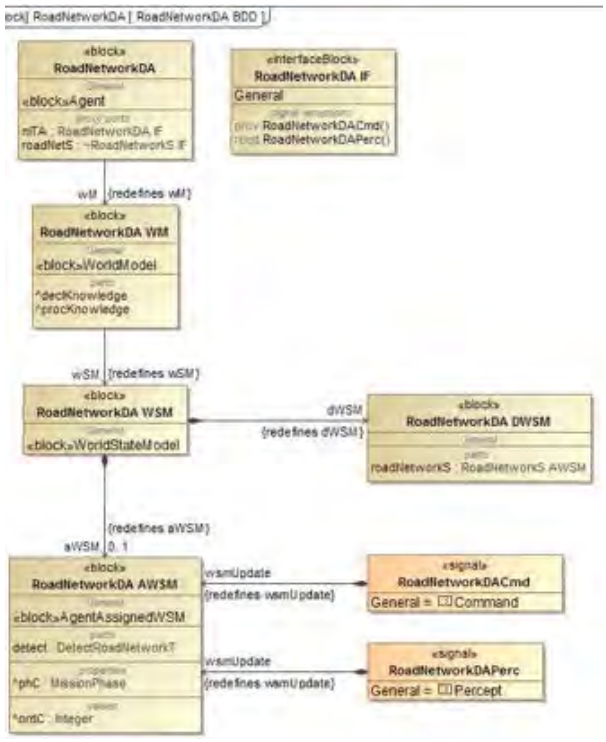

Figure 184. Mobility Road Network DA Internal Composition

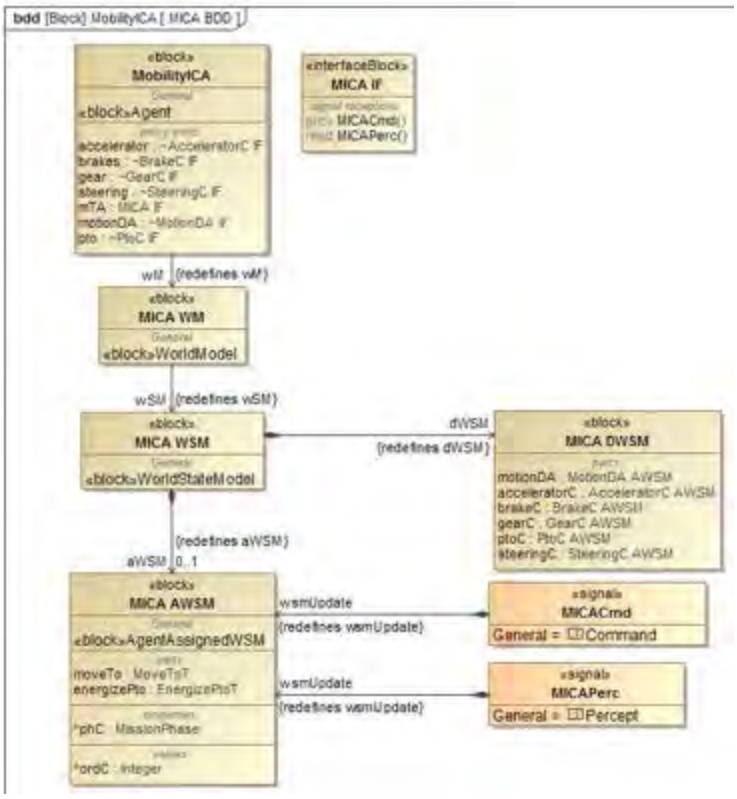

Figure 185. Mobility ICA Internal Composition

5. Mobility Agent Behavior

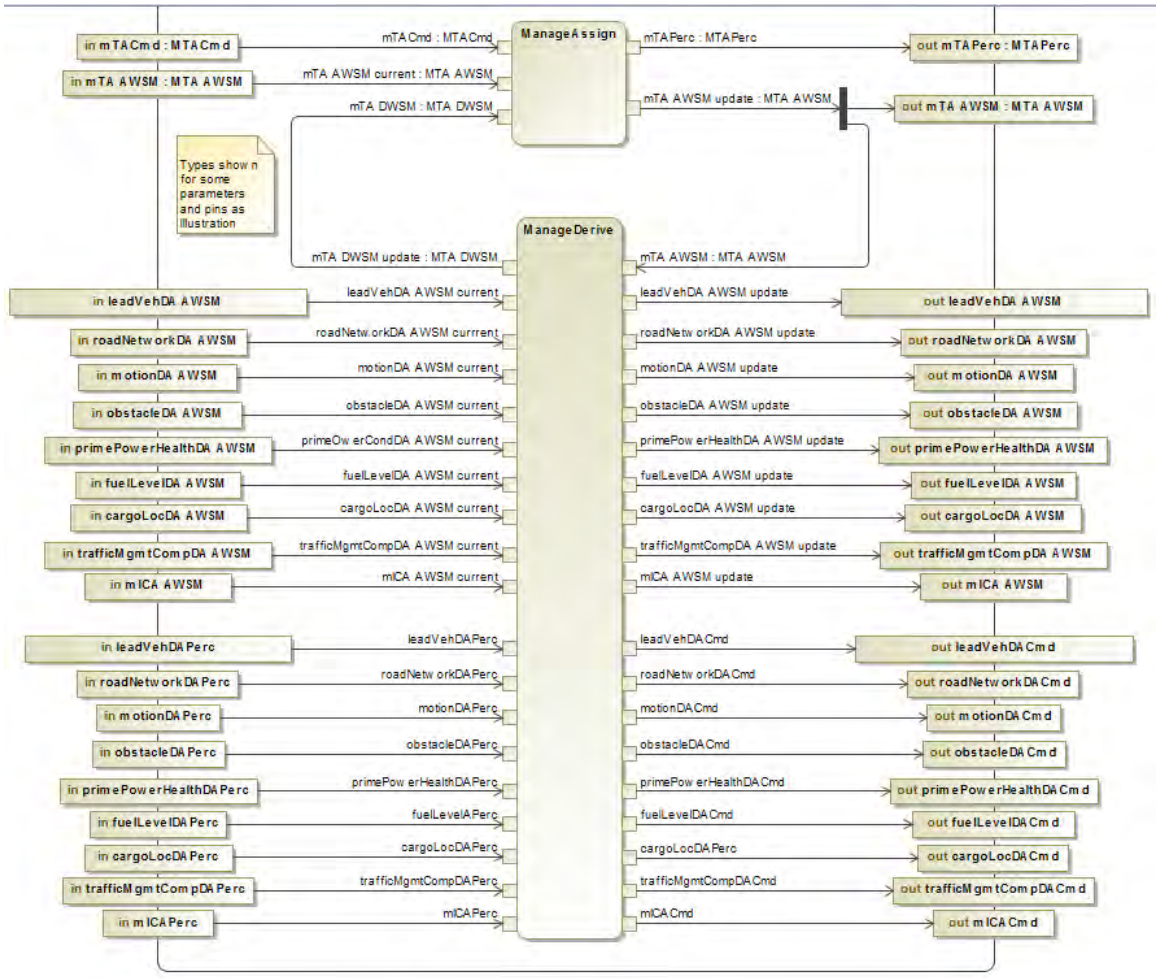

Figure 186. Mobility Task Agent Behavior

F. SYSTEM COMMAND CONTROL AND AUTONOMICS

System Command Control and Autonomics (SC2A)

1. SC2A Assigned Mission

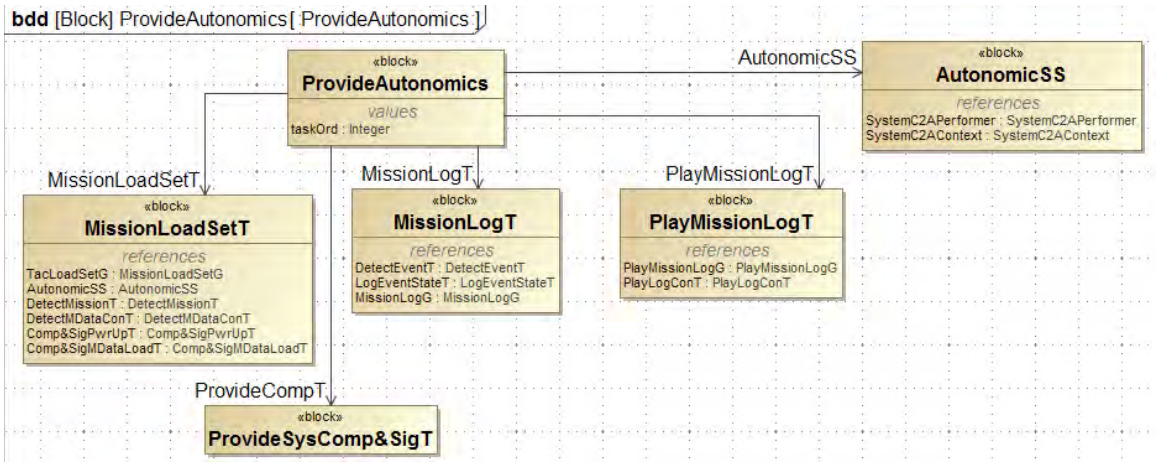

Figure 187. SC2A Provide Autonomics Task

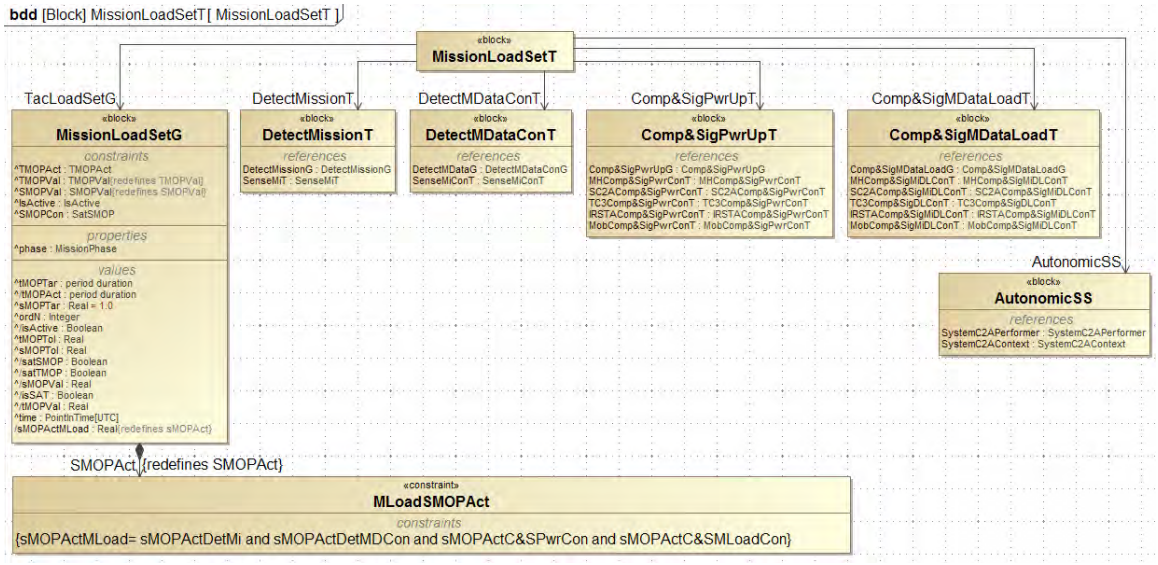

Figure 188. SC2A Mission Load Set Desired Trajectory

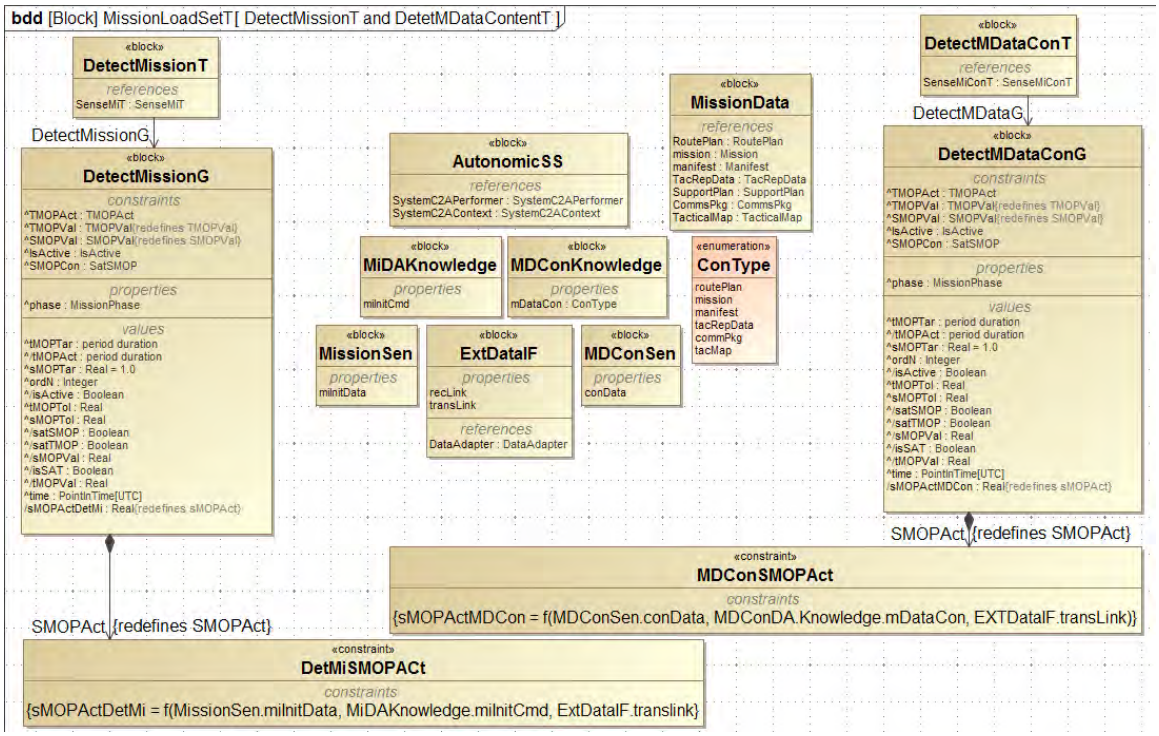

Figure 189. SC2A Detect Mission and Detect Mission Data Content Desired Trajectories

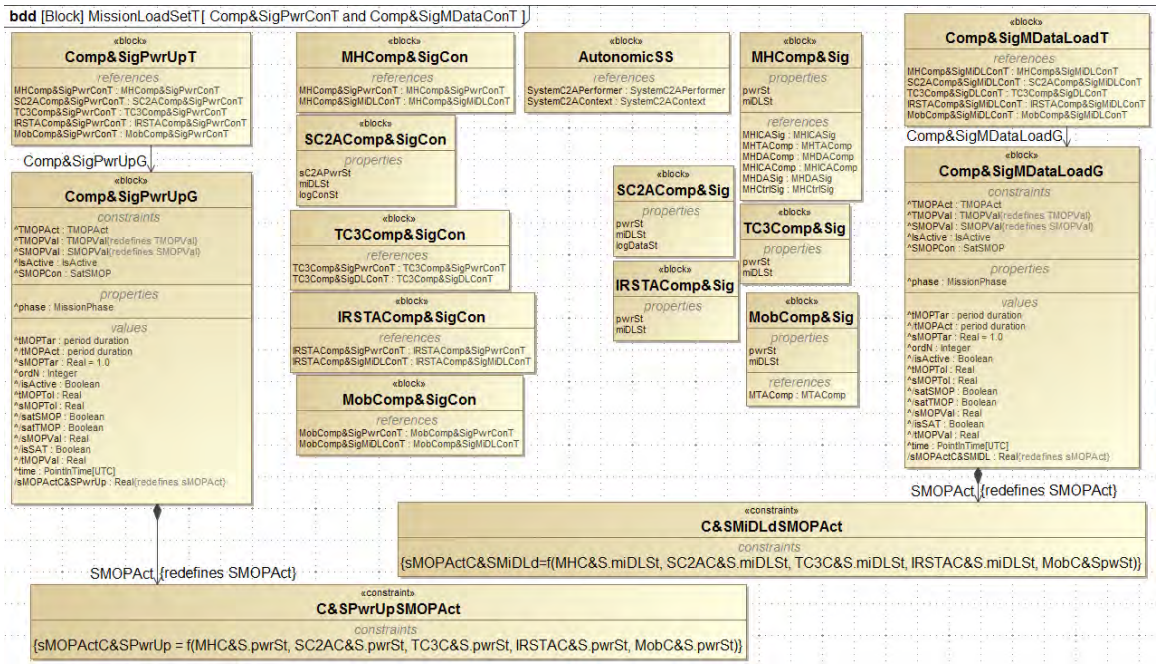

Figure 190. SC2A Computation & Signal Power Up and Data Load Desired Trajectories

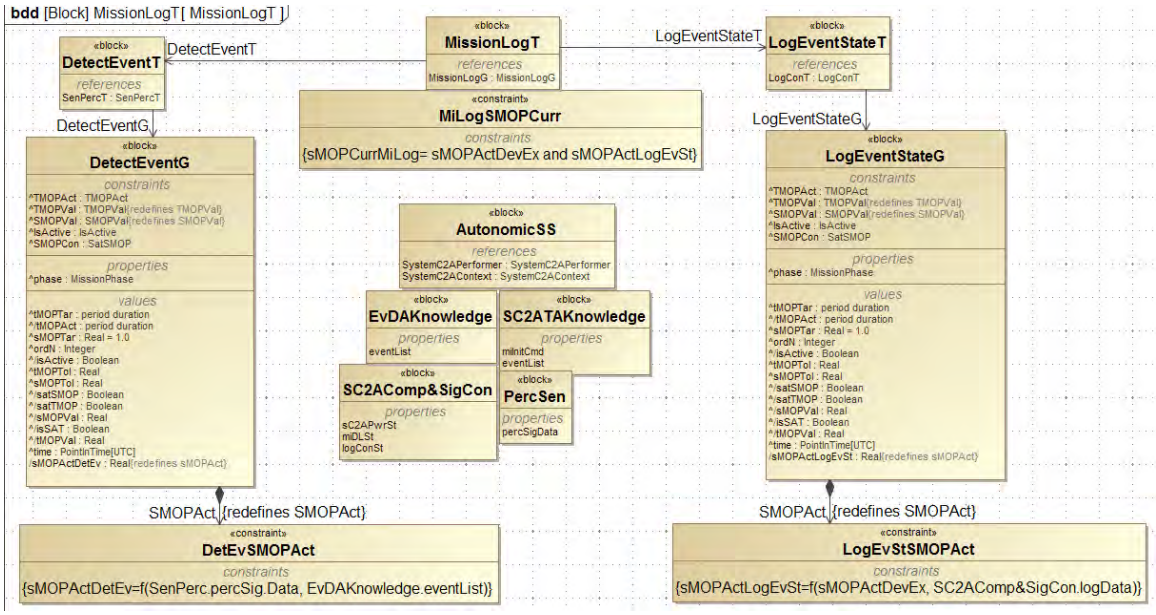

Figure 191. SC2A Mission Log Desired Trajectories

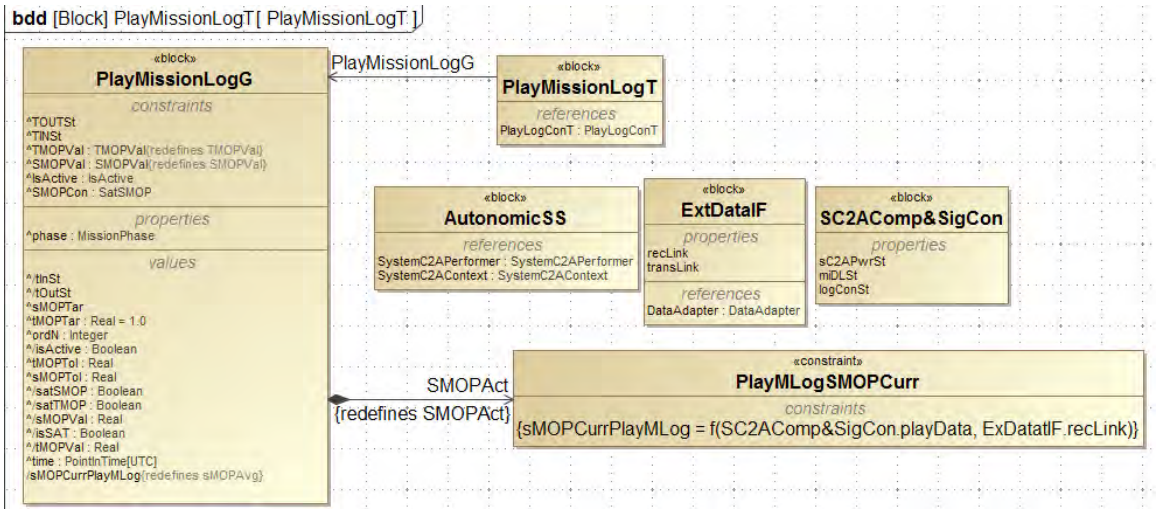

Figure 192. SC2A Play Mission Log Desired Trajectory

2. SC2A Logical Object Hierarchy

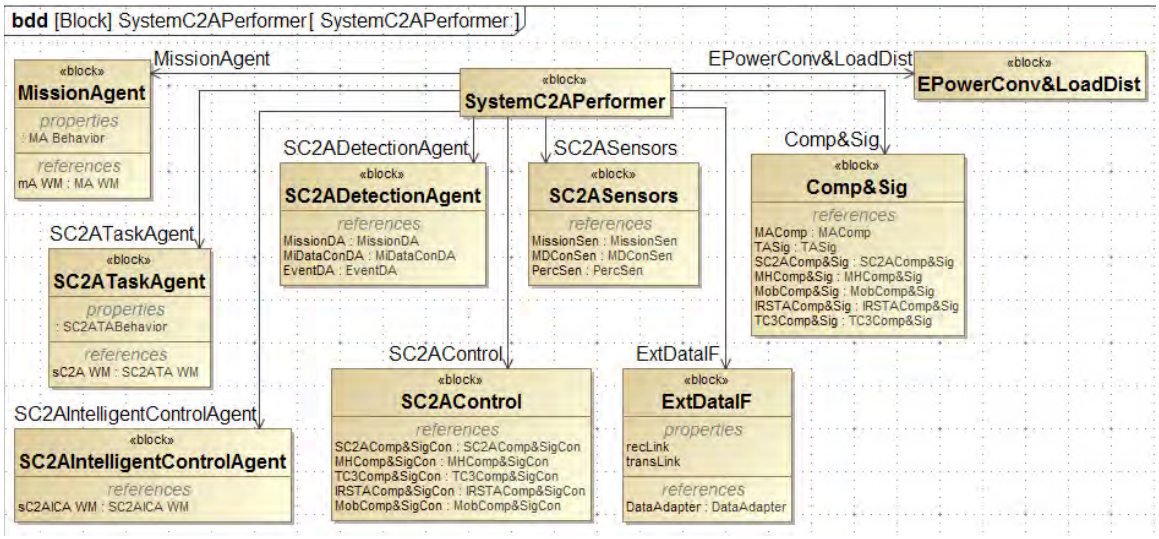

Figure 193. SC2A Performer Logical Objects

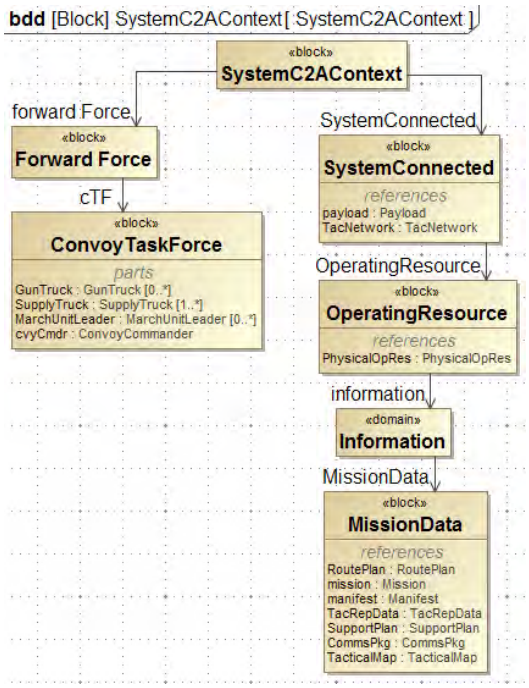

Figure 194. SC2A Context Logical Objects

3. SC2A Interactions

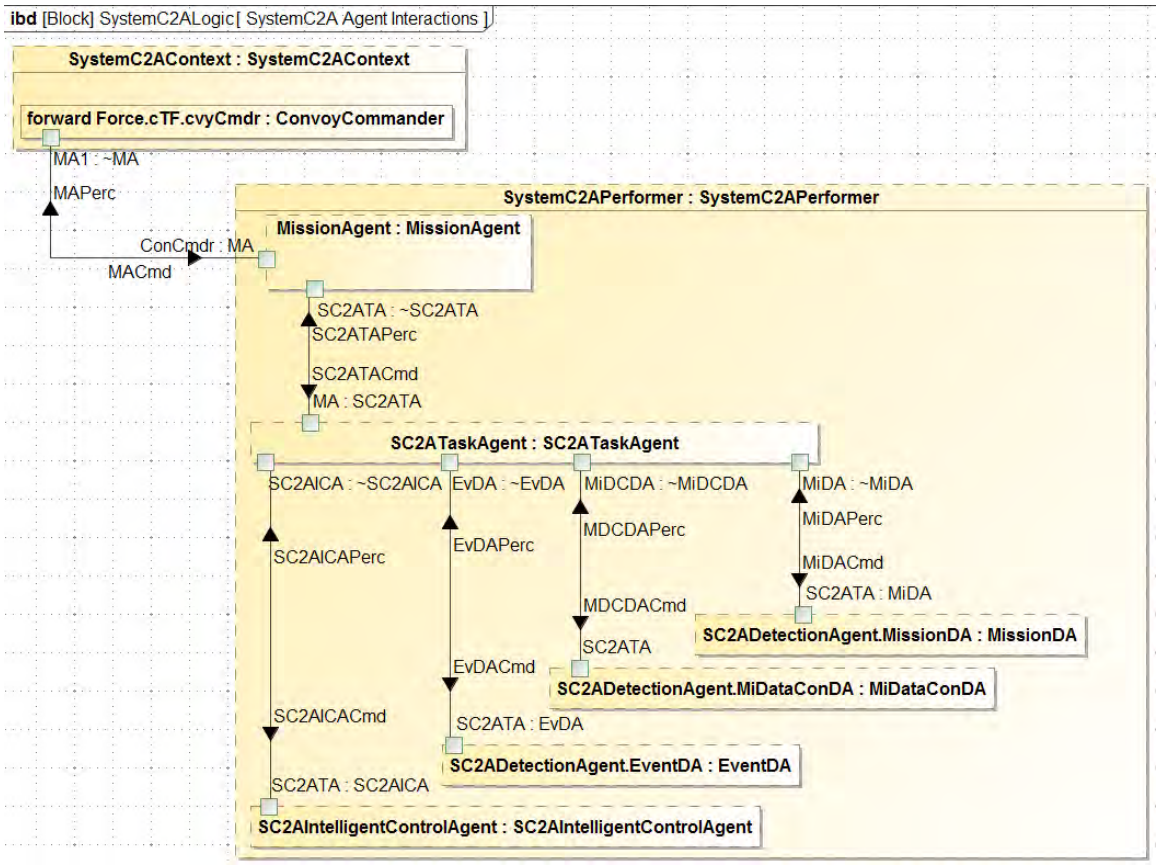

Figure 195. SC2A Agent and Mission Agent/Convoy Commander Interactions

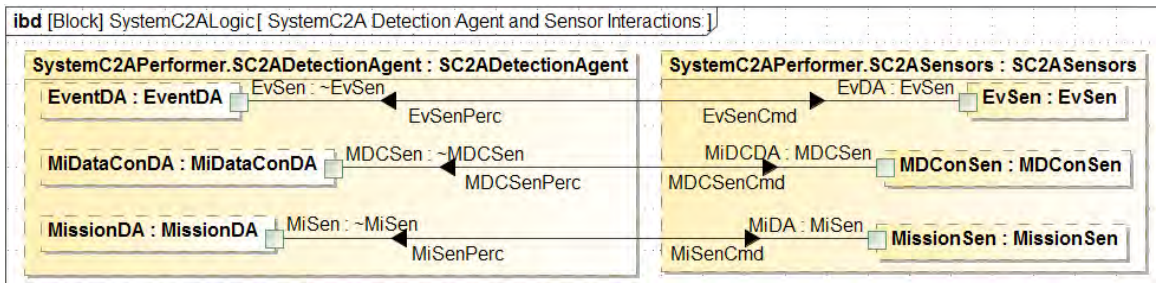

Figure 196. SC2A Detection Agent and Sensor Interactions

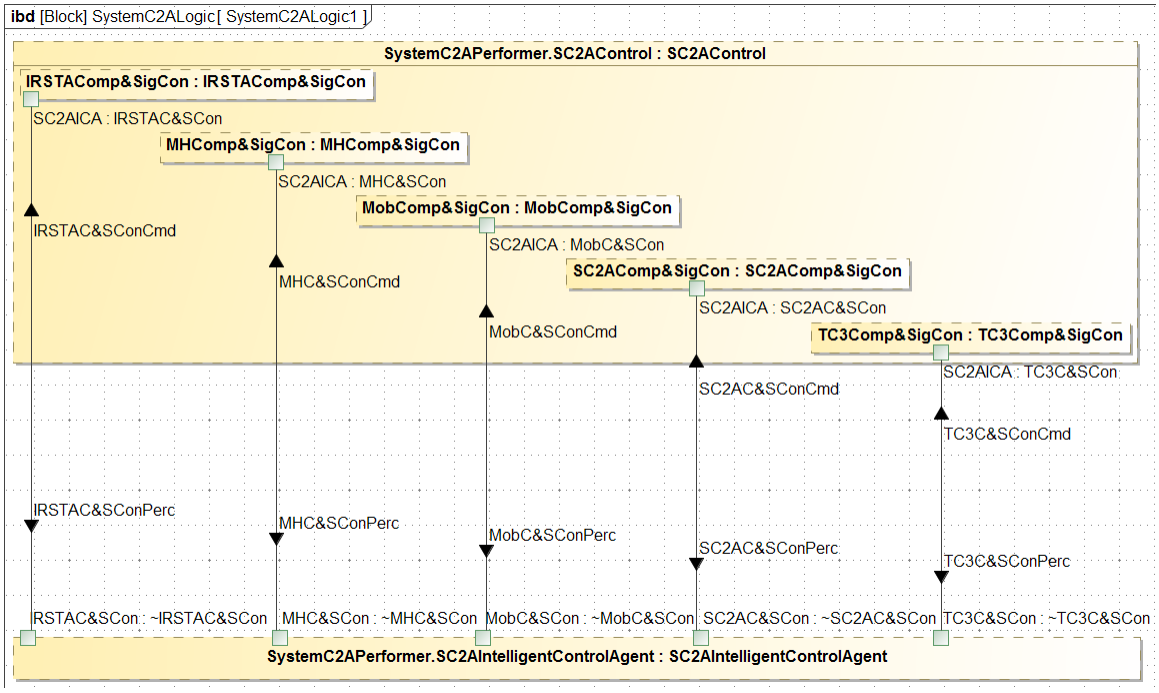

Figure 197. SC2A Intelligent Control and “Controller” Interactions

4. SC2A Agent Internal Composition

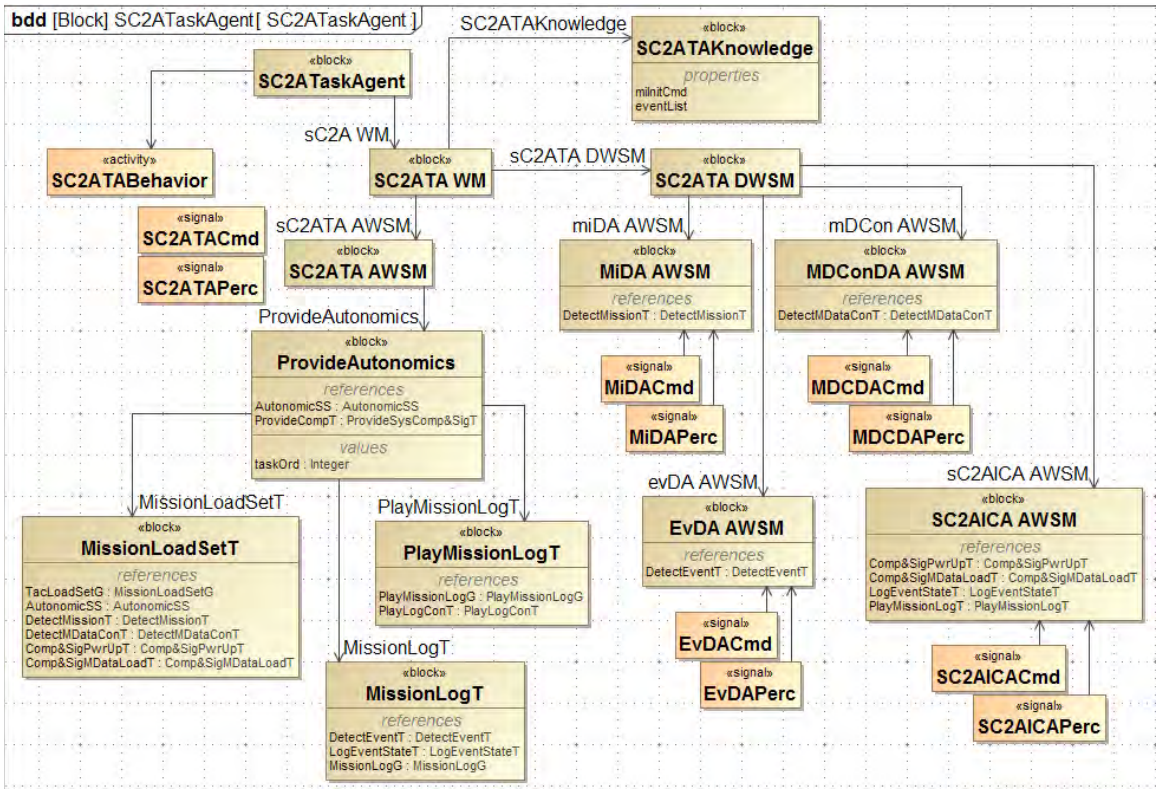

Figure 198. SC2A Task Agent Internal Composition

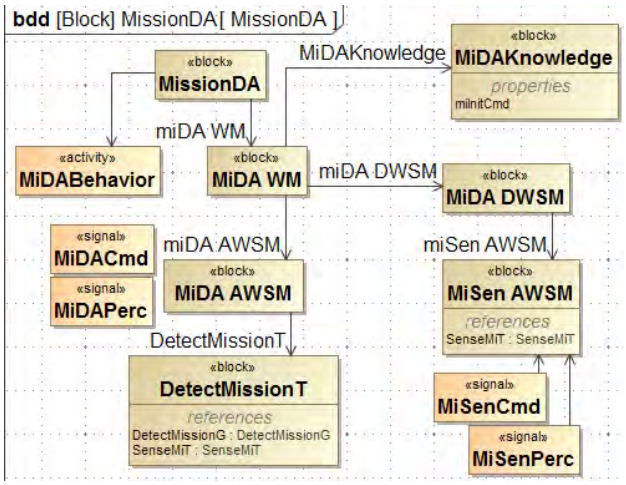

Figure 199. SC2A Mission DA Internal Composition

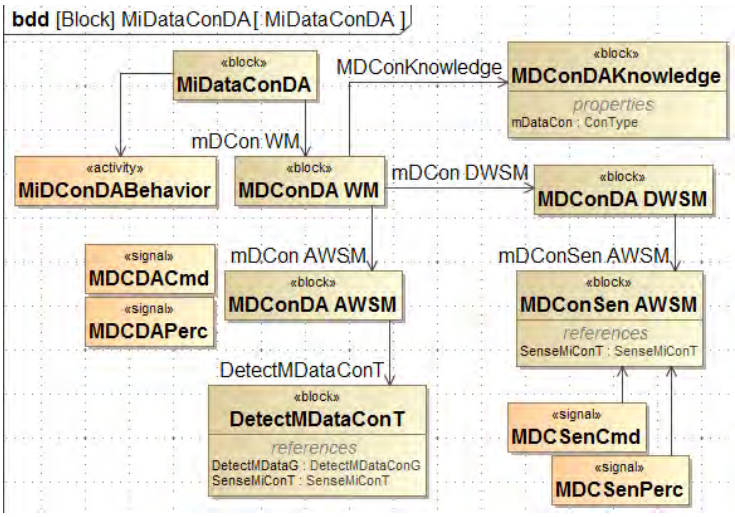

Figure 200. SC2A Mission Data Content DA Internal Composition

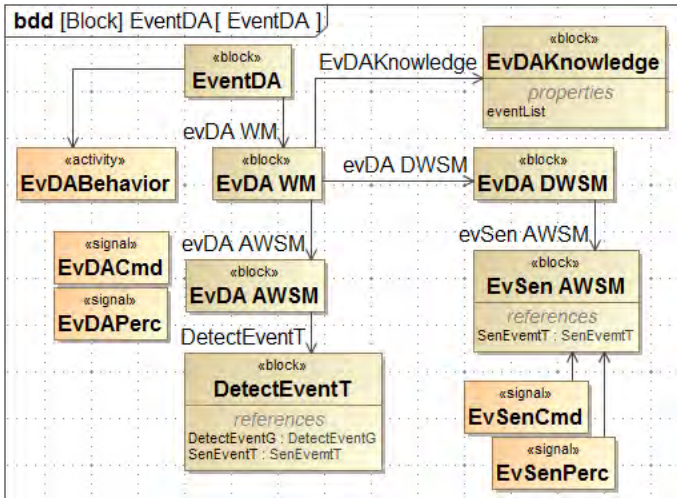

Figure 201. SC2A Event DA Internal Composition

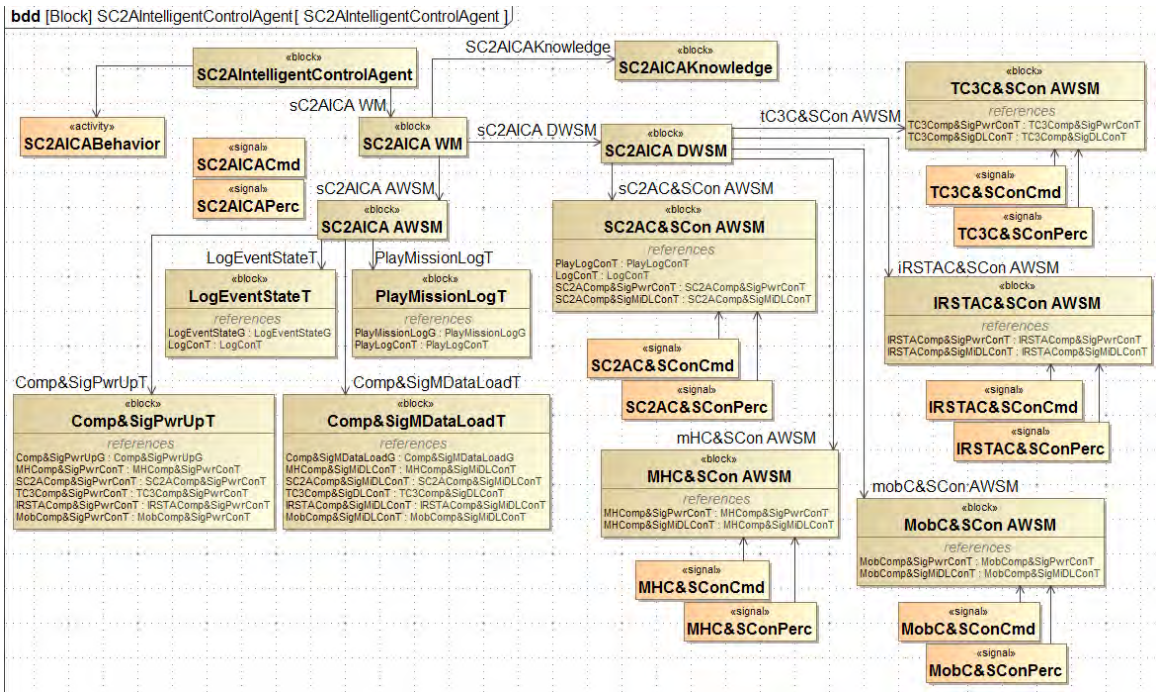

Figure 202. SC2A Intelligent Control Agent Internal Composition

5. SC2A Agent Behavior

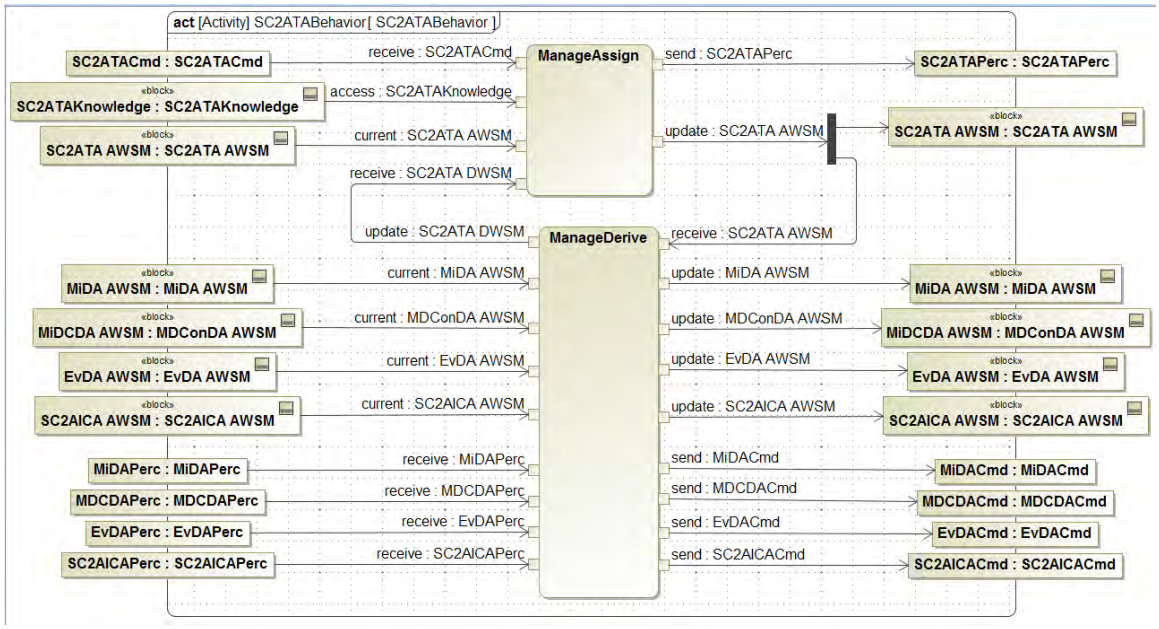

Figure 203. SC2A Task Agent Behavior

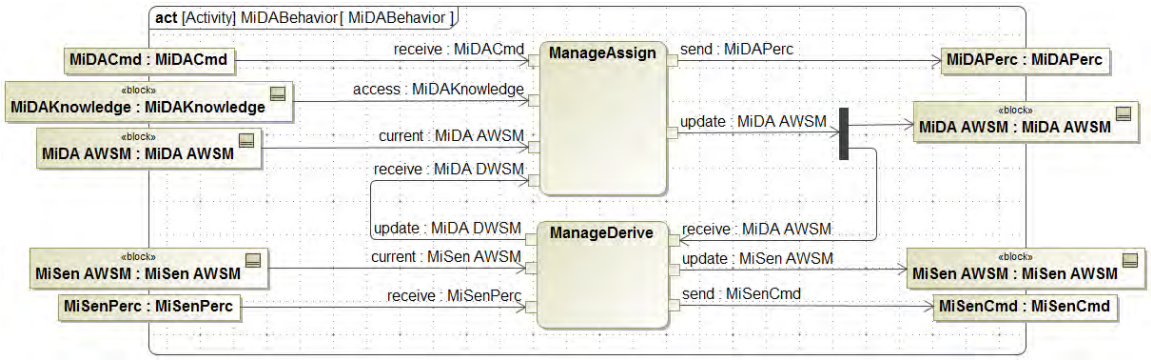

Figure 204. SC2A Mission DA Behavior

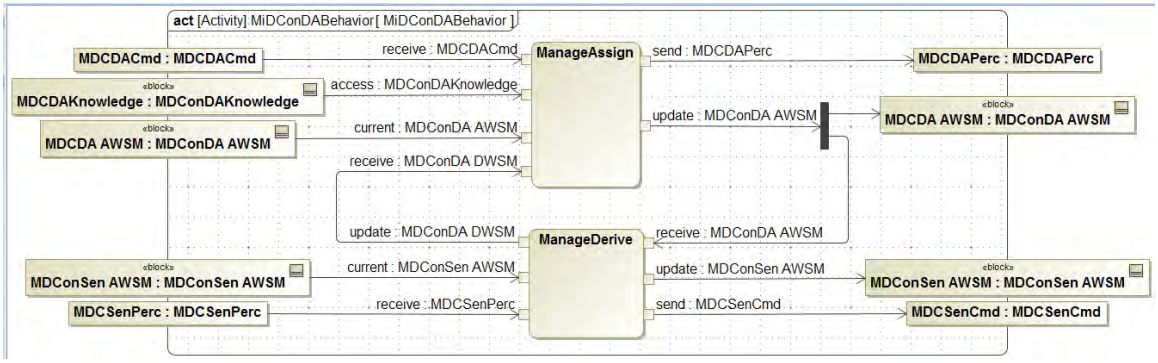

Figure 205. SC2A Mission Data Content DA Behavior

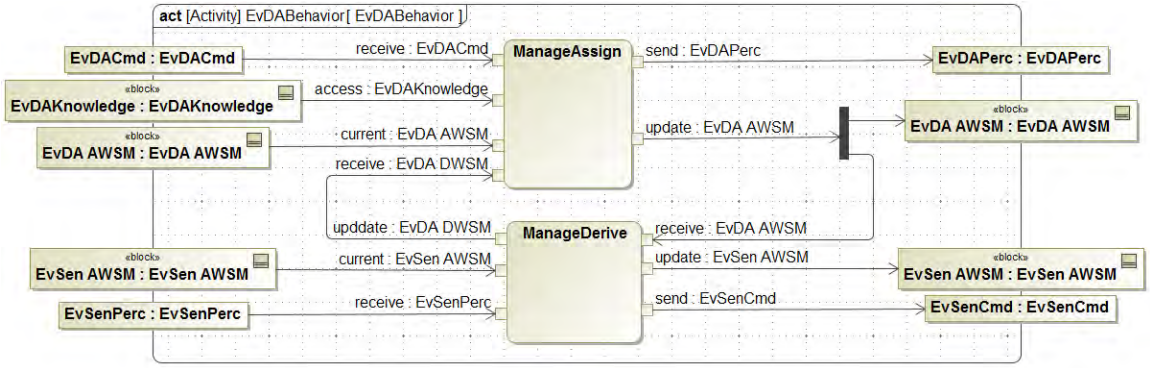

Figure 206. SC2A Event DA Behavior

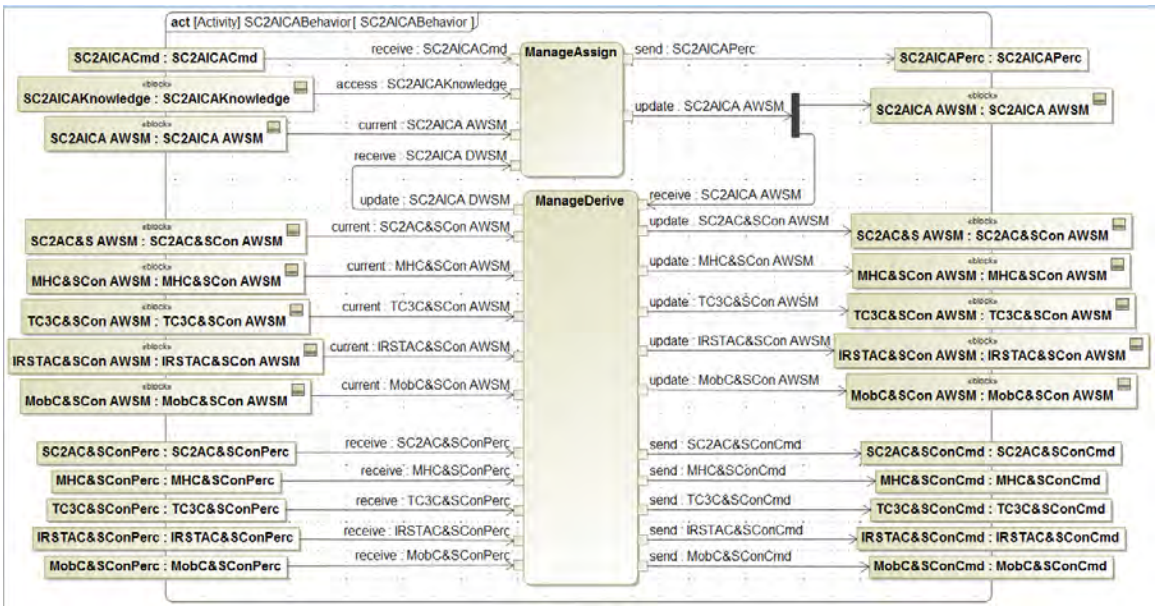

Figure 207. SC2A Intelligent Control Agent Behavior

LIST OF REFERENCES

Ackoff, Russell L. 1989. "From Data to Wisdom," *Journal of Applied System Analysis*, 16: 3–9.

Agha, Gul. 1986. *MIT Press Series in Artificial Intelligence*. Cambridge, MA: MIT Press.

Ahmadi, Hossein, Tarek Abdelzaher, Jiawei Han, Nam Pham, and Raghu K. Ganti. 2011. "The Sparse Regression Cube: A Reliable Modeling Technique for Open Cyber-Physical Systems," In *IEEE/ACM Second International Conference on Cyber-Physical Systems* 87–96. DOI: 10.1109/ICCPS.2011.20.

Albus, James S. 2002. "4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles," *Proceedings of the SPIE 16th Annual Symposium on Aerospace/Defense Sensing, Simulation and Controls* 4715Orlando, FL. http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=821736.

Albus, James S., and Alexander M. Meystel. 2001. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. *Wiley Series on Intelligent Systems*, edited by James S. Albus, Alexander M. Meystel, and Lofti Zadeh. New York: John Wiley and Sons.

Albus, James S., and Anthony J. Barbera. 2004, "RCS: A Cognitive Architecture for Intelligent Multi-Agent Systems," *Proceedings of 5th International Federation of Automatic Control (IFAC)/EURON Symposium on Intelligent Autonomous Vehicles Proceedings*, edited by Jose Santo-Victor and M. Isabel Ribeiro, 29 (1): 87–99. Lisbon, Portugal: Published for the International Federation of Automatic Control by Elsevier, Oxford University, 2005.

Altintas, Ilkay, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," *Proceedings of the 16th International Conference on Scientific and Statistical Database Management* 423–424. New York: IEEE Computer Society. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9176.

Al-Hammouri, Ahmad T. 2012, "A comprehensive co-simulation platform for cyber-physical systems," Computer Communications 36, no.1 (Dec): 8–19.

Argente, Estefania, Vicente Julian, and Vicente Botti. 2006. "Multi-Agent System Development Based on Organizations," *Electronic Notes in Theoretical Computer Science*, 150 (3): 55–71.

Arthur, W. Brian. 2009. *The Nature of Technology*. New York: Free Press, A Division of Simon and Schuster.

Arunachalam, M, R Arun Prakash, and R Rajesh. 2014. "A Typical Approach in Conceptual and Embodiment Design of Foldable Bicycle," *International Journal of Computer Applications*, 87, no.19 (Feb): 0975—8887.

Aziz, Muhammad Waqar, Muhammad Rashid. 2016, "Domain Specific Modeling Language for Cyber-physical Systems," *18th International Conference on Information Systems Engineering Proceedings*, ICISE 2016, 29–33, Los Angeles, California, IEEE Computer Society, July, https://www.computer.org/csdl/proceedings/icise/2016/2287/00/index.html.

Badger, Michael, Dennis Bushmitch, Rick Cozby, John Hartwig, and Brian Hobson. 2013, "Network Capability Assessment and Standardized Measures of Performance (MOP) Framework," *The ITEA Journal of Test and Evaluation*, 34, no.4 (December).

Baheti, Radhakisan, and Helen Gill. 2011. "Cyber-physical Systems," *The Impact of Control Technology*, 12: 161–166.

Barbacci, Mario, Mark H. Klein, Thomas A. Longstaff, and Charles B. Weinstock. 1995, *Quality Attributes*, Technical Report CMU/SEI-95-TR-021, Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/TechnicalReport/1995_005_001_16427.pdf.

Behjati, Razieh, Tao Yue, Shiva Nejati, Lionel Briand, and Bran Selic. 2011, "Extending SysML with AADL Concepts for Comprehensive System Architecture Modeling," *Modeling Foundations and Applications 7th European Conference Proceedings, ECMFA 2011*, edited by Robert France et al. Lecture Notes in Computer Science, 6698: 236–252. Berlin, Germany: Springer.

Bergenti, Federico, and Agostino Poggi. 2000, "Exploiting UML in the Design of Multi-Agent Systems," *Proceedings of the ECOOP Workshop on Engineering Societies in the Agents World 2000 (ESAW 00)*, edited by Andrea Omicini, Robert Tolksdorf, and Franco Zambonelli, 1: 96–103, Berlin, Germany: Springer.

Bhave, Ajinkya, Bruce Krogh, David Garlan, and Bradley Schmerl. 2010, *Multi-domain Modeling of Cyber-Physical Systems Using Architecture Views*, Institute for Software Research Carnegie Mellon University, Report Number 11–2010 http://repository.cmu.edu/cgi/viewcontent.cgi?article=2013&context=isr.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 2011. *Systems Engineering and Analysis*, Fifth Edition, New Jersey: Prentice Hall.

Booch, Grady, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, and Kelli A. Houston. 2007. *Object-Oriented Analysis and Design with Applications*, Third Edition, pp22-23, Massachusetts: Pearson, Addison-Wesley Object Technology Series.

Bures, Tomas, Mark Klein, Danny Weyns, and Rodolfo E. Haber. 2015, "1st International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS 2015)," *37th IEEE International Conference on Software Engineering Proceedings*, edited by Richard L. Wexelblat et al. 2: 1009–1010, Florence/Firenze Italy, IEEE Press.

Bures, Tomas, Petr Hnetynka, Jan Kofron, Rima Al Ali, and Dominik Skoda. 2016, "Statistical Approach to Architecture Modes in Smart Cyber-physical Systems," *2016 13th Working IEEE/IFIP Conference on Software Architecture*, 1: 168–182, Venice Italy, IEEE Computing Society Conference Publication Services.

Chief Information Officer, U.S. Department of Defense, DOD Architecture Framework Version 2.02. 2010 "DODAF Formal Ontology," http://DODcio.defense.gov/Library/DODArchitectureFramework/DODaf20_ontology1.aspx.

Delgoshaei, Parastoo, Mark A. Austin, and Amanda J. Pertzborn. 2014. "A Semantic Framework for Modeling and Simulation of Cyber-Physical Systems," *International Journal on Advances in Systems and Measurements* 7(3&4): 223–238.

Deloach, Scott A., Mark F. Wood, and Clint H. Sparkman. 2001. "Multiagent Systems Engineering," *International Journal of Software Engineering and Knowledge Engineering*, 11 no. 3 (June): 231–258.

Department of the Army. 2004. *Operations Terms and Graphics*, FM 1–02. Washington, DC: Department of the Army. http://www.bits.de/NRANEU/others/amd-us-archive/adrp1-02%282-15%29.pdf.

Department of the Army. 2008. *Operations*, FM 3–0. Washington, DC: Department of the Army. http://www.bits.de/NRANEU/others/amd-us-archive/adrp3-0%2801%29.pdf.

Department of the Army.2012. *Army Universal Tasks List*, FM 7–15. Washington, DC: Department of the Army. http://www.bits.de/NRANEU/others/amd-us-archive/adrp7-15%2803%29.pdf.

Department of the Navy. 2015. Systems Engineering Technical Review Process. NAVAIR Instruction 4355.19E. Patuxent River, MD. http://www.navair.navy.mil/nawctsd/Resources/Library/Acqguide/NAVAIRINST-4355-19.pdf.

Derler, Patricia, Edward A. Lee, and Alberto L. Sangiovanni-Vincentelli. 2011, *Addressing Modeling Challenges in Cyber-Physical Systems*, Technical Report No. UCB/EECS-2011-17, Electrical Engineering and Computer Sciences, University of California at Berkeley, www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-17.html.

Ding, Li, Pranam Kolari, Zhongli Ding, Sasikanth Avancha, Tim Finin, and Anupam Joshi. 2005, *Using Ontologies in the Semantic Web: A Survey*, TR CS-05-07, Department of Computer Science and Electrical Engineering, University of Maryland, July, downloaded from http://ebiquity.umbc.edu/_file_directory_/papers/209.pdf, Aug 23, 2016.

Dori, Dov. 2014, *The Maturation of Model Based Systems Engineering: OPM as the ISO Conceptual Modeling Language Standard*, Massachusetts Institute of Technology System Design and Management Webinar, June 2. https://sdm.mit.edu/news/news_articles/webinar_060214/webinar060214.pdf.

Douglas, Bruce Powell. 2004. *Real Time UML: Advances in the UML for Real-Time Systems*. 3rd ed. Boston MA: Addison-Wesley Object-technology Series, Pearson Education Inc.

Estefan, Jeff A. 2008, "Survey of Model Based Systems Engineering (MBSE) Methodologies," Revision B, INCOSE MBSE Initiative. http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf.

Evans, John M., Elena R. Messina, James S. Albus, and Craig I. Schlenoff, 2002. "Knowledge Engineering for Real Time Intelligent Control," *Proceedings of the International Workshop on Intelligent Knowledge Management Techniques (I—KOMAT 2002)*, Crema, Italy, Sept. 16–18.

Fichman, Robert G., and Chris F. Kemerer. 1992. "Object Oriented and Conventional Analysis and Design Methodologies," *Computer*, 25 (10): 22–39.

Fritzson, Peter. 2012, *Principles of Object Oriented Modeling and Simulation with Modelica*, Open Source Modelica Publication Tutorial, downloaded from www.openmodelica.org on Aug 13, 2016.

Garlan, David, Robert T. Monroe, and David Wile. 2000. "Acme: Architectural Description of Component-Based Systems," Foundations of Component Based Systems, edited by Gary T. Leavens and Murali Sitaraman, 68: 47–68, New York, NY, Cambridge University Press.

Giachetti, Ronald E. 2010. *Design of Enterprise Systems: Theory, Architecture and Methods*. Florida: CRC Press, pp 60–62.

Giachetti, Ronald E. 2015. "Evaluation of the DODAF's Meta Model's Support of Systems Engineering," *Complex Adaptive Systems Publication 5, Procedia Computer Science* 61: 254–260.

Goedicke M. 1990. "Paradigms of modular system development," Chapter 1, *Managing Complexity in Software Engineering,* Edited by Dr. R.J. Mitchell, Peter Peregrinus Ltd. On behalf of the Institution of Electrical Engineers.

Graja, Imen, Slim Kallel, Nawal Geurmouche, and Ahmed Hadj Kacem. 2016. "BPMN4CPS: A BPMN extension for modeling Cyber-Physical Systems," *25th IEEE Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises WETICE-2016*, edited by Sumitra M. Reddy and Walid Gaaloul 1: 152–157. Paris France, IEEE Computer Society Conference Publishing Services.

Gunes, Volkan, Steffen Peter, Tony Givargis, and Frank Vahid. 2014. "A Survey on Concepts, Applications and Challenges in Cyber-Physical Systems," *KSII Transactions on Internet and Information Systems* 8 no. 12 (December): 4242–4268. http://www.itiis.org/digital-library/manuscript/894, doi.10.3837/tiis.2014.12.001.

Hart, Laura E. 2015, *Introduction to Model-Based Systems Engineering and SysML*, Delaware Valley Chapter INCOSE Meeting, presentation copyright Lockheed Martin Corporation. https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf.

Hause, Matthew, and Paul Pearce. 2012. "ISO-15288, OOSEM and Model-Based Submarine Design." SETE APCOSE 2012. http://www.omgsysml.org/Pearce_Hause_ISO-15288_OOSEM_and_Model-Based_Submarine_Design_SETE_APCOSE_20121.pdf.

Henry, Stephen M., Lucas A. Waddell, and Mike R. DiNunzio. 2016. "The Whole System Trades Analysis Tool for Autonomous Ground Systems," *2016 NDIA Ground Vehicle Systems Engineering and Technology Symposium*, Aug 2–4.

Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sidhu Ram, 2004. "Decision Science in Information Systems Research," *Management Information Systems Quarterly* 28(1): 75–105.

Hewitt, Carl. 1977, "Viewing control structures as patterns of passing messages," *Journal of Artificial Intelligence* 8(3): 323–364.

Hoffmann, Hans-Peter Ph.D. 2011. *Systems Engineering Best Practices with the Rational Solution for Systems and Software Engineering Deskbook Release 3.1.2*, IBM Software Group. http://www-01.ibm.com/support/docview.wss?uid=swg27023356&aid=1.

Hudak, John, and Peter Feiler. 2007. *Developing AADL Models for Control Systems: A Practitioner's Guide*, Software Engineering Institute, Technical Report CMU/SEI-2007-TR-014, ESC-TR-2007-014. https://resources.sei.cmu.edu/asset_files/TechnicalReport/2007_005_001_14891.pdf.

Iivari, Juhani. 2007. "A Paradigmatic Analysis of Information Systems as a Design Science," *Scandinavian Journal of Information Systems* 19(2): 39–64.

Iivari, Juhani. 2015, "Distinguishing and Contrasting Two Strategies for Design Science Research," *European Journal of Information Systems* 24(1): 107–115.

JC3IEDM. 2007, *The Joint C3 Information Exchange Data Model Metamodel Version 3.1 Annex G2*, NATO Multilateral Interoperability Programme.

Jensen, Jeff C., Danica H. Chang, and Edward A. Lee. 2011. "A Model-Based Design Methodology for Cyber-Physical Systems," *Proceedings First IEEE Workshop on Design, Modeling and Evaluation of Cyber-Physical Systems (CyPhy)* 1: 1666–1671.

Jha, Susmit, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. "Synthesizing Switching Logic for Safety and Dwell-Time Requirements," *Proceedings of the 1st IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS 2010)* 1: 22–31.

Jirkovsky, Vaclav, Marek Obitko, and Vladimir Marik. 2017. "Understanding Data Heterogeneity in the Context of Cyber-Physical Systems Integration." *IEEE Transactions on Industrial Informatics* 13(2): 660–667.

Jobst, Martin Erich, and Christian Prehofer. 2016. "Towards Hierarchical Information Architectures in Automotive Systems," *3rd International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC)*, Conference location Vienna Austria. DOI: 10.1109/EITEC.2016.7503695.

Khaitan, Siddhartha Kumar, and James D. McCalley. 2015. "Design Techniques and Applications of Cyberphysical Systems: A Survey," *IEEE Systems Journal* 9(2): 350–365.

Kruchten, Phillippe. 2004. *The Rational Unified Process: An Introduction* 3rd Edition, Massachusetts: Pearson, Addison-Wesley Professional.

Kruger, K., and A.H. Basson. 2013. "Multi-agent Systems vs IEC 61499 for Holonic Resource Control in Reconfigurable Systems," *Forty Sixth CIRP Conference on Manufacturing Systems Procedia CIRP*, edited by Pedro F. Cunha, 7: 503–508. https://www.sciencedirect.com/journal/procedia-cirp/vol/7.

INCOSE. 2007. *INCOSE SE Vision 2020*, Technical Report INCOSE-TP-2004-004-02. http://sebokwiki.org/wiki/INCOSE_Systems_Engineering_Vision_2020.

Lee, Edward A. 2003, "Model-Driven Development—From Object-Oriented Design to Actor-Oriented Design," *Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (The Monterey Workshop)*, Chicago. September 24, 2003. https://www.cs.uic.edu/~shatz/SEES/Schedule.htm.

Lee, Edward A. 2008. *Cyber-Physical Systems: Design Challenges*, Technical Report Number UCB/EECS-2008-8, January 23, downloaded from http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-8.html .

Lee, Edward A., and Sanjit A. Seshia. 2017. *Introduction to Embedded Systems, A Cyber-Physical System Approach*, Second Edition. Cambridge, MA: MIT Press,

Lee, Edward A., and Stephen Neuendorffer. 2004. Classes and Subclasses in Actor-Oriented Design," *Conference on Formal Methods and Models for Codesign (MEMCODE)*, San Diego, CA. June 22–25, 2004. https://ptolemy.eecs.berkeley.edu/publications/papers/04/Classes/Lee_Classes.pdf .

Lee, Jay, Behrad Bagheri, and Hung-An Kao. 2014. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters* 3: 18–23.

Leitao, Paulo, Vladimir Marik, and Pavel Vrba. 2013. "Past, Present, and Future of Industrial Agent Applications," *IEEE Transactions on Industrial Informatics* 9(4): 2360–2372.

Loos, Sarah M., Andre Platzer, and Ligia Nistor. 2011. "Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified," *Proceedings 17th International Symposium of Formal Methods*, edited by Michael Butler and Wolfram Schulte, 42–56. Springer http://www.springer.com/la/book/9783642214363.

Lukasiewycz, Martin, Sebastian Steinhorst, Florian Sagstetter, Wanli Chang, Peter Waszecki, Matthias Kauer, and Samarjit Chakraborty. 2012. "Cyber-Physical Systems Design for Electric Vehicles," *15th Euromicro Conference on Digital System Design (Cesme Ismer, Turkey)*, 477–484 http://ieeexplore.ieee.org/abstract/document/6386930/.

Lynch, Kevin, Randall Ramsey, George Ball, Dale B. Larkin, Matt Schmit, Kyle Collins, Justin Knight, Ted Bapty, Jason Scott. 2016, "Ontology-Driven Metamodel Validation in Cyber-Physical Systems," *AIAA Modeling and Simulation Technologies Conference 2016*, AIAA Aviation Forum. https://doi.org/10.2514/6.2016-4005.

Ma, Zhiqiang, Xiao Fu, and Zhenhua Yu. 2012. "Object-Oriented Petri Nets Based Formal Modeling for High-Confidence Cyber-Physical Systems," *8th International Conference on Wireless Communications, Networking and Mobile Computing WiCOM 2012*. DOI: 10.1109/WiCOM.2012.6478590.

Malan, Ruth, and Dana Bredemeyer. 2001. *Functional Requirements and Use Cases*, Bredemeyer Consulting, White Paper, http://www.bredemeyer.com/pdf_files/functreq.pdf.

Magureanu, Gabriela, Madalin Gavrilescu, Dan Pescaru, and Alex Doboli. 2010. "Towards UML Modeling of Cyber-Physical System: A Case Study for Gas Distribution," *Proceedings 8th IEEE International Symposium on Intelligent Systems and Informatics*. doi:10.1109/SISY.2010.5647314.

Mayk, Israel, and William C. Regli (Editors). 2006. *Agent Systems Reference Model*, Technical Report for DOD Contract #DAAB07-01-9-L504: Version 1.0a. Drexel University, Camden NJ. http://www.fipa.org/docs/ACIN-reference_model-v1a.pdf.

Merson, Paulo. 2009. *Data Model as an Architectural View*, Technical Note CMU/SEI-2009-TN-024, Carnegie Mellon Software Engineering Institute, http://repository.cmu.edu/sei/285/.

MIL-STD-881C, 2011, *Department of Defense Standard Practice Work Breakdown Structures for Defense Materiel Items,* AMSC 9213, October 3.

Neema, Sandeep, Jason Scott, and Ted Bapty. 2015. *CyPhyML Language in the META Toolchain*, Technical Report ISIS-15-104, Institute for Software-Integrated Systems, Vanderbilt University https://pdfs.semanticscholar.org/e075/648ed1c76580ba3c53cf090c5cc778cd06ac.pdf.

NDIA. 2011. *Final Report Model Based Engineering (MBE)*, Subcommittee, Jeff Bergenthal (Subcommittee Lead), NDIA Systems Engineering Division M&S Division. https://www.ndia.org/-/media/sites/ndia/meetings-and-events/3187-sullivan/divisions/systems-engineering/modeling-and-simulation/reports/model-based-engineering.ashx.

No Magic Inc. 2015, *Magic Draw Users Manual*, Version 18.1, No Magic Inc., page 673, Figure 436.

NGAUS, 2014, NGAUS (National Guard Association of the United States) FY2014 Fact Sheet, downloaded on Mar23 2016 from http://www.ngaus.org/sites/default/files/AbramsFactSheetFY14.pdf.

NIST. 2013, *Foundations for Innovation in Cyber-Physical Systems*, Workshop Report, prepared by Energetics Inc for National Institute for Standards and Technology, Columbia, MD https://www.nist.gov/sites/default/files/documents/el/CPS-WorkshopReport-1-30-13-Final.pdf.

NSF 2015. *Cyber-Physical System Vision Statement Working Draft*, National Science Foundation, Directorate for Computer & Information Science & Engineering, https://www.nitrd.gov/nitrdgroups/images/6/6a/Cyber_Physical_Systems_(CPS)_Vision_Statement.pdf.

O'Brien, Liam, Len Bass, and Paulo Merson. 2005. *Quality Attributes and Service-Oriented Architectures*, Technical Note CMU/SEI-2005-TN-014, Software Engineering Institute, Carnegie Mellon University. http://repository.cmu.edu/sei/449/.

Odell, James, H. Van Dyke Parunak, and Bernhard Bauer. 2000. "Extending UML for Agents," *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 3–17. https://www.aaai.org/Press/Proceedings/aaai00.php.

OMG MBSE Wiki. 2011,"INCOSE Object Oriented Systems Engineering Methodology (OOSEM)," Last modified April 25 http://www.omgwiki.org/MBSE/doku.php?id=mbse:incoseoosem.

Pang, Cheng, Wenbin Dai, and Valeriy Vyatkin. 2015. "Towards IEC 61499 Models of Computation in Ptolemy II," *Industrial Electronics Society IECON2015, 41st Annual Conference*, 1988–1993 DOI:10.1109/IECON.2015.7392392.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. 2008. "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems* 24 no. 3 (Winter): 45–77.

Poole, David, and Alan Mackworth. 2010. *Artificial Intelligence: Foundations of Computational Agents*, Cambridge University Press, accessed from http://artint.info/index.html.

Ptolemy II, 2014, *System Design, Modeling, and Simulation Using Ptolemy II*, First Edition, Version 1.0.2, Ptolemaeus, Claudius (Editor), Ptolemy.org, http://ptolemy.org/systems.

Rajhans, Akshay, Ajinkya Bhave, Ivan Ruchkin, Bruce H. Krogh, David Garlan, Andre Platzer, and Bradley Schmerl. 2014. "Supporting Heterogeneity for Cyber-Physical Systems Architectures," *IEEE Transactions on Automatic Control* 59, no. 12 (December) 3178–3193.

Rational 1998. *Rational Unified Process Best Practices for Software Development Teams*, Rational Software White Paper, TP026B, Rev 11/01, Rational Software Corporation. https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf.

Regli, William C., Israel Mayk, Christopher T. Cannon, Joseph B. Kopena, Robert N. Lass, William M. Mongan, Duc N. Nguyen, Jeff K. Salvage, Evan A. Sultanik, and Kyle Usbeck. 2014. "Development and Specification of a Reference Architecture for Agent-Based Systems," *IEEE Transactions on Systems, Man and Cybernetics: Systems* 44 (2): 146–161.

Riff, Richard. 2010, *A Global PLM strategy in Ford Motor Company*, Collaboration & Interoperability Congress, May3-5, downloaded from www.mcadcafe.com, on Aug 21, 2016.

Ross, Adam, and Daniel E. Hastings. 2005. "The Tradespace Exploration Paradigm," *INCOSE International Symposium*, 15 (1): 21–35. http://onlinelibrary.wiley.com/doi/10.1002/iis2.2005.15.issue-1/issuetoc.

Ross, Douglas T. 1977. "Structured Analysis: A Language for Communicating Ideas," *IEEE Transactions on Software Engineering* SE-3 no. 1 (January): 16–34.

Russell, Stuart, and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach*. 2nd ed. Upper Saddle River, NJ: Prentice Hall Series in Artificial Intelligence, Pearson Education Inc.

Rzevski, George. 2003, "On conceptual design of intelligent mechatronic systems," *Mechatronics* 13 (10): 1029–1044.

Salazar, Luis A. Cruz, Oscar A. Rojas Alvarado. 2014, "The Future of Industrial Automation and IEC 61499 Standard," *2014 3rd International Congress of Engineering Mechatronics and Automation (CIIMA)*, Cartagena, Columbia. 10.1109/CIIMA.2014.6983434.

Sanchez, Jose L. Fernandez. 2012, *An Integrated Systems and Software Engineering Process (ISE&PPOOA)*, Autonomous Systems Laboratory (ASLab), Technical University of Madrid http://tierra.aslab.upm.es/public/index.php?option=com_content&task=category&sectionid=6&id=19&Itemid=41.

Sangiovanni-Vincentelli, Alberto. 2008. "Is a Unified Methodology for System-Level Design Possible?" *IEEE Design & Test of Computers* 25 (4): 346–357.

Sasidharan, Swaytha, Andrey Somov, Abdur Rahim Biswas, and Raffaele Giaffreda. 2014. "Cognitive Management Framework for Internet of Things—A Prototype Implementation," *2014 IEEE World Forum on Internet of Things (WF IoT) Proceedings* 538–543. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6784568.

Schmerl, Bradley, and David Garlan. 2004. "AcmeStudio: Supporting Style-Centered Architecture Development," *Proceedings of the 26th International Conference on Software Engineering* 704–705. Edinburgh, Scotland. DOI: 10.1109/ICSE.2004.1317401.

Seepersad, Carolyn C., Kjartan Pedersen, Jan Emblemsvag, Reid Bailey, Janet. K. Allen, and Farrokh Mistree. 2005. *The Validation Square: How Does One Verify and Validate a Design Method*, ASME Press, New York, NY, 2005 https://www.me.utexas.edu/~ppmdlab/files/ccs.valid.square.Jan05.pdf.

SEI (Software Engineering Institute). 2016, SEI Product Line Bibliography, White Paper, Carnegie Mellon University, https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=513798.

Semy, Salim K., Pulvermacher, Mary K., and Orbst, Leo J. 2004. *Toward the Use of an Upper Ontology for U.S. Government and U.S. Military Domains: An Evaluation*, MITRE Technical Report, Document Number MTR 04B0000063. https://www.mitre.org/publications/technical-papers/toward-the-use-of-an-upper-ontology-for-us-government-and-us-military-domains-an-evaluation.

Shames, Peter M., and Marc A. Sarrel. 2015. "A Modeling Pattern for Layered System Interfaces," *25th Annual INCOSE International Symposium (IS2015)*, Seattle, WA. July 13–16.

Shoham, Yoav. 1993. "Agent-oriented programming," *Artificial Intelligence* 60 (1): 51–92.

Simon, Herbert A. 1962. "The Architecture of Complexity," *Proceedings of the American Philosophical Society* 106 no. 6 (December): 467–82.

Smith, Barry, Waclaw Kusnierczyk, Daniel Schober, and Werner Ceusters. 2006. "Towards a Reference Terminology for Ontology Research and Development in the Biomedical Domain," *Proceedings for KR-MED 2006 Biomedical Ontology in Action*, 222: 57–65, Baltimore, MD: CEUR.

.Sorouri, Majid, Sandeep Patil, Zoran Salcic, and Valeriy Vyatkin. 2015. "Software Composition and Distributed Operation Scheduling in Modular Automated Machines," *IEEE Transactions on Industrial Informatics*, 11 no. 4 (August): 865–878. DOI: 10.1109/TII.2015.2430836.

Spero, Eric, Michael Avera, Pierre Valdez, and Simon Goerger. 2015. *Tradespace Exploration for the Engineering of Resilient Systems*, Technical Report ARL-TR-7288, U.S. Army Research Laboratory, Defense Technical Information Center, May.

Stumpf, Ondrej, Tomas Bures, and Vladimir Matena. 2015. "Security and Trust in Data Sharing Smart Cyber-Physical Systems," *Proceedings of the European Conference on Software Architecture Workshops*, edited by Ivica Cmkovic, Dubrovnik/Cavtat, Croatia. DOI: http://doi.acm.org/10.1145/2797433.2797451.

Tatikonda, M.V., and U. Wemmerlov. 1992. "Adoption and Implementation of group technology classification and coding systems; insights from seven case studies," *Internal Journal of Production Research*, 30 (9): 2087–2110.

Thramboulidis, Kleanthis. 2010. "The 3+1 SysML View-Model in Model Integrated Mechatronics," *Journal Software Engineering & Applications* 3 (February): 109–118. www.SciRP.org/journal/jsea.

Topper, J. Stephen, and Nathaniel C. Horner. 2013. *Model-Based Systems Engineering in Support of Complex Systems Development*, John Hopkins APL Technical Digest, Volume 32, Number 1. http://techdigest.jhuapl.edu/td/td3201/32_01-topper.pdf.

Tveit, Amund. 2001. "A Survey of Agent-Oriented Software Engineering," *Computer Science Graduate Student Conference (SCGSC)*, Norwegian University of Science and Technology (NTNU). http://csgsc.idi.ntnu.no/2001/pages/papers/atveit.pdf.

Van Brussel, Hendrik, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. 1998. "Reference architecture for holonic manufacturing systems: PROSA," *Computers in Industry* 37 (3): 255–274.

Van Ruijven, L.C. 2013. "Ontology for Systems Engineering," *Procedia Computer Science* 160: 383–392. Conference on System Engineering Research, CSER'13. https://ac.els-cdn.com/S1877050913000410/1-s2.0-S1877050913000410-main.pdf?_tid=cbb7ed82-8eb1-4e4e-a926-df8e34a5243d&acdnat=1520115885_6be4d1ce9391033b64b0a7a86ec6f843.

Vyatkin, Valeriy, Cheng Pang, and Stavros Tripakis. 2015. "Towards Cyber-physical Agnosticism by Enhancing IEC 61499 with PTIDES Models of Computation," *41st Annual Conference of the IEEE Industrial Electronics Society (IECON2015)* 01970-01975. http://ieeexplore.ieee.org/document/7392389/.

Wagh, Aditya, Xu Li, Jingyan Wan, Chunming Qiao, and Changxu Wu. 2011. "Human Centric Data Fusion in Vehicular Cyber-Physical Systems," *Proceedings IEEE First International Workshop on Cyber-Physical Networking Systems* 695–700. http://cse.unl.edu/~byrav/INFOCOM2011/workshops/papers/p695-wagh.pdf DOI: 10.1109/INFCOMW.2011.5928763.

Wagner, David A., Matthew B. Bennett, Robert Karban, Nicolas Rouquette, Steven Jenkins, and Michel Ingham. 2012. "An Ontology for State Analysis: Formalizing the Mapping to SysML." *Aerospace Conference, 2012 IEEE*, 1–16. Big Sky, MT. DOI: 10.1109/AERO.2012.6187335.

Wan, Jiang, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. 2017. "Cyber-Physical Codesign at the Functional Level for Multidomain Automotive Systems," *IEEE Systems Journal* 11 (4): 2949–2959.

Willems, Jan C. 2007. "The behavioral approach to open and interconnected systems," *IEEE Control Systems Magazine*, 27 no. 6 (December): 46–99.

Woodbridge, Michael, Nicholas R. Jennings, and David Kinny. 2000. "The Gaia Methodology for Agent-Oriented Analysis and Design," *Journal Autonomous Agents and Multi-Agent Systems* 3 no. 3 (September): 285–312.

Zimmerman, Hubert. 1980. “OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection,” *IEEE Transactions on Communications* Com-28 no. 4 (April): 425–432.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California